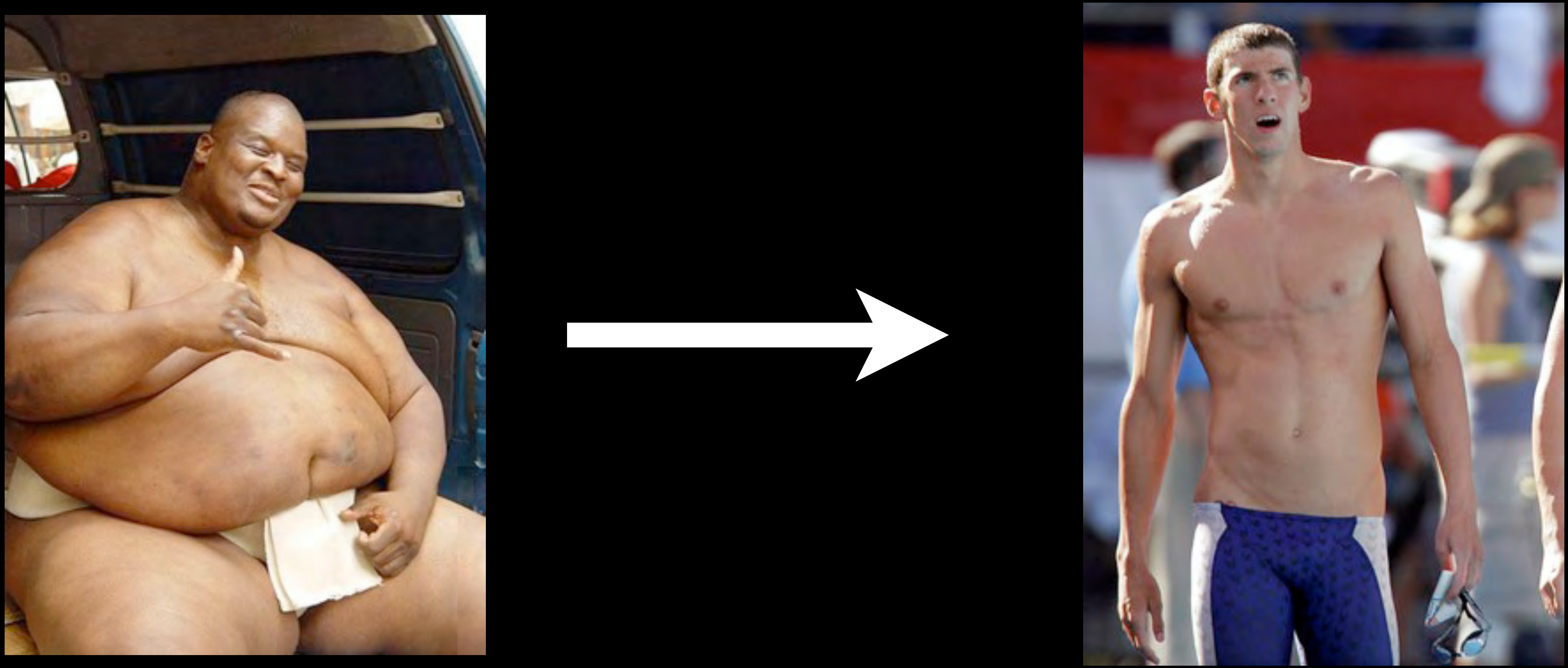
function hooking for osx and linux

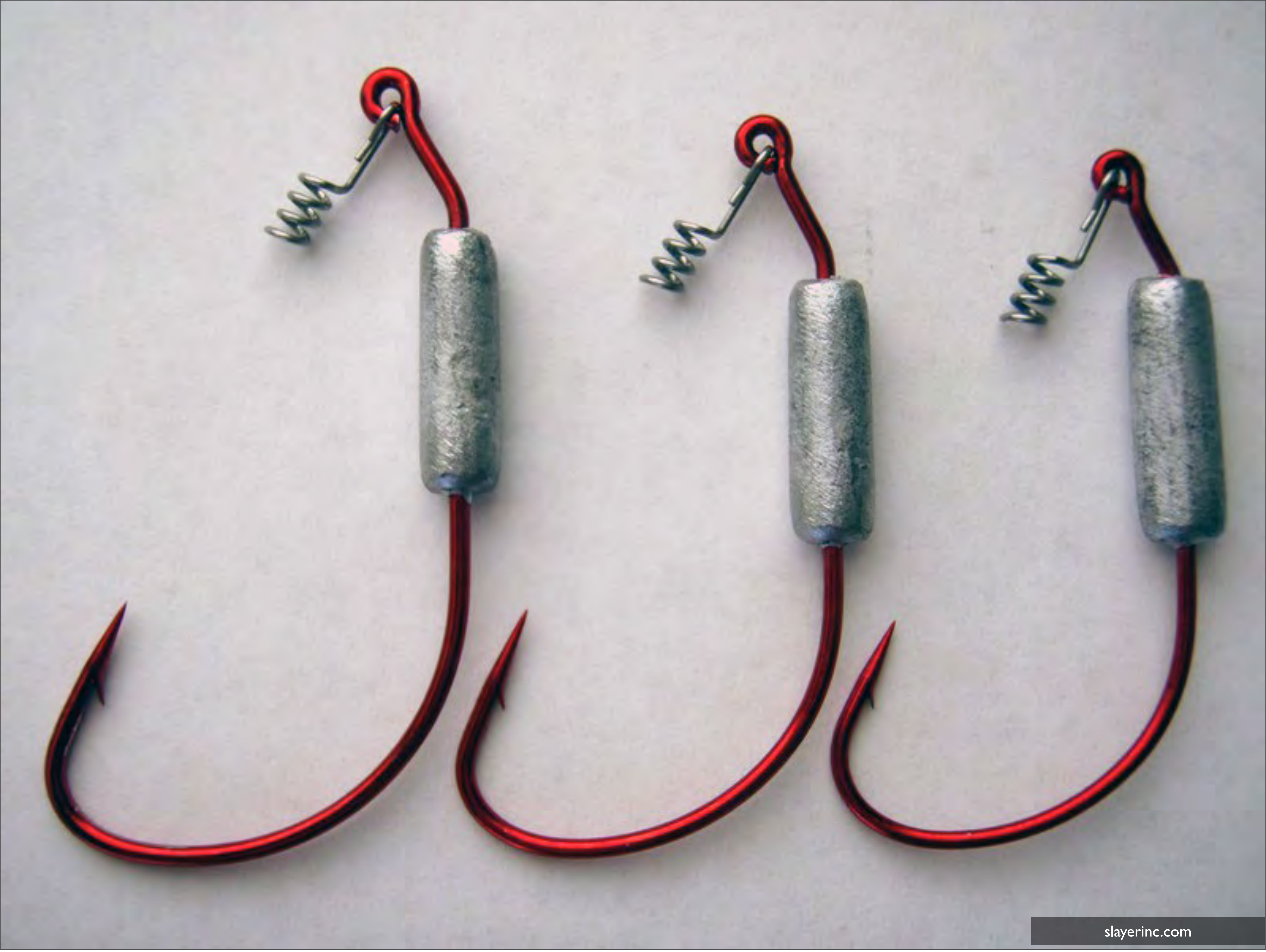
joe damato
@joedamato
timetobleed.com

slides on timetobleed.com

call me a script kiddie:
@joedamato



globalhealthandfitness.com



slayerinc.com

assembly is in att syntax

dbgrady.files.wordpress.com

WTF is an ABI ?

WTF is an Application Binary Interface ?

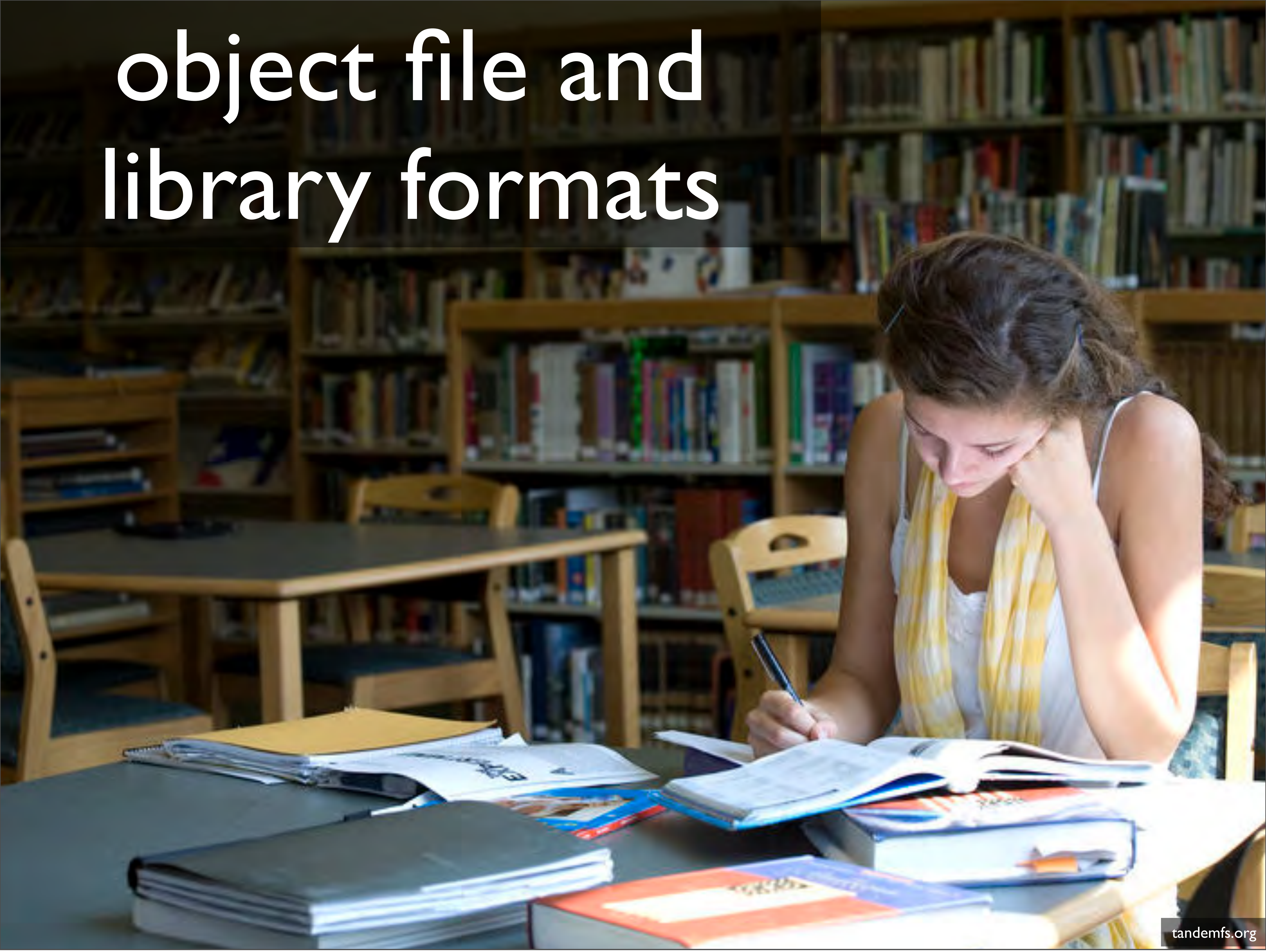
alignment

thomasgroup.com

calling convention

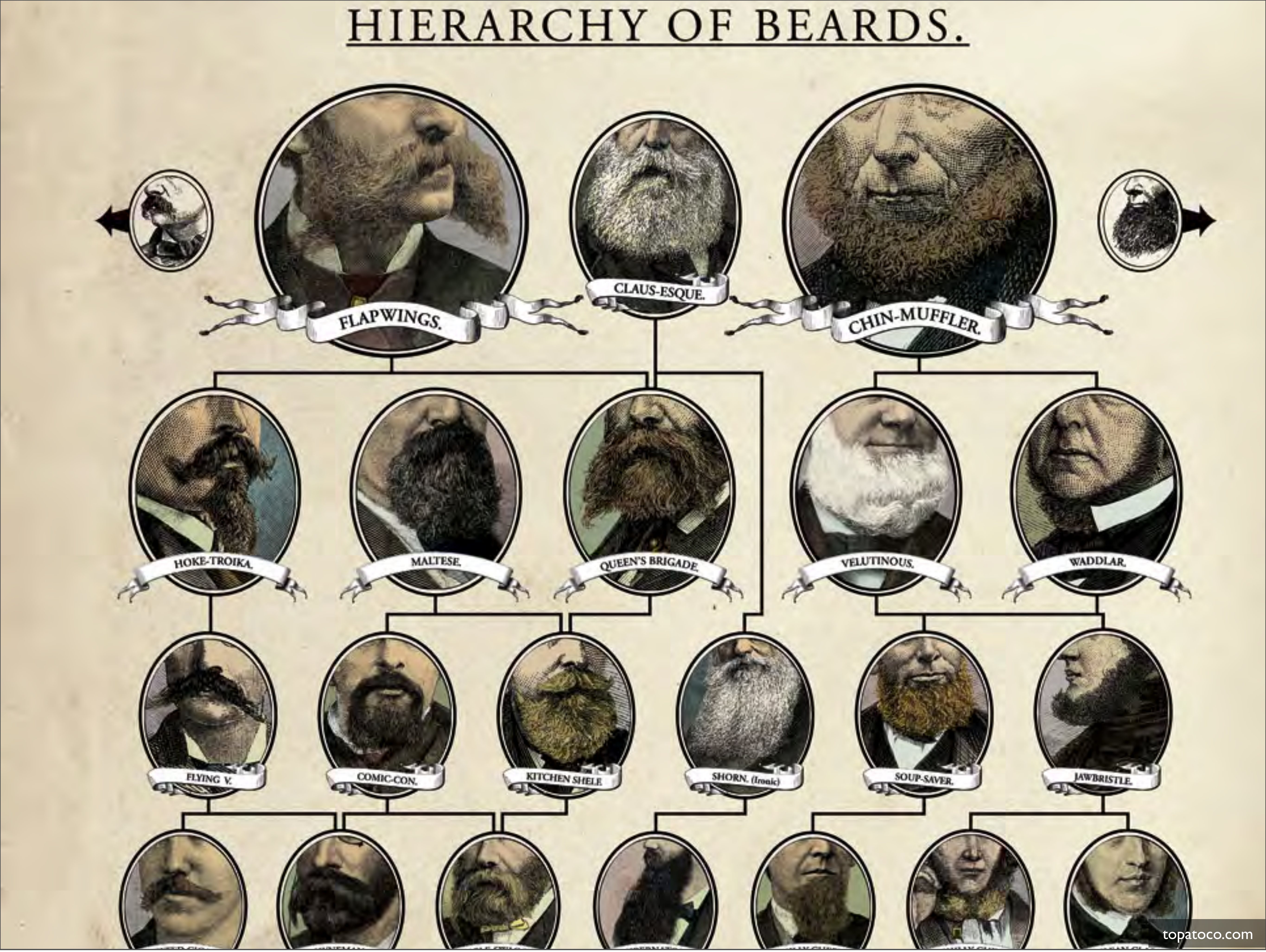
arianlim.wordpress.com

object file and library formats



tandemfs.org

hierarchy of specs


HIERARCHY OF BEARDS.
FLAPWINGS.
CLAUS-ESQUE.
CHIN-MUFFLER.
HOKE-TROIKA.
MALTESE.
QUEEN'S BRIGADE.
VELUTINOUS.
WADDLAR.
FLYING V.
COMIC-CON.
KITCHEN SHELF.
SHORN. (Ironic)
SOUP-SAVER.
JAWBRISTLE.
topatoco.com

System V ABI (271 pages)

System V ABI AMD64 Architecture Processor Supplement (128 pages)

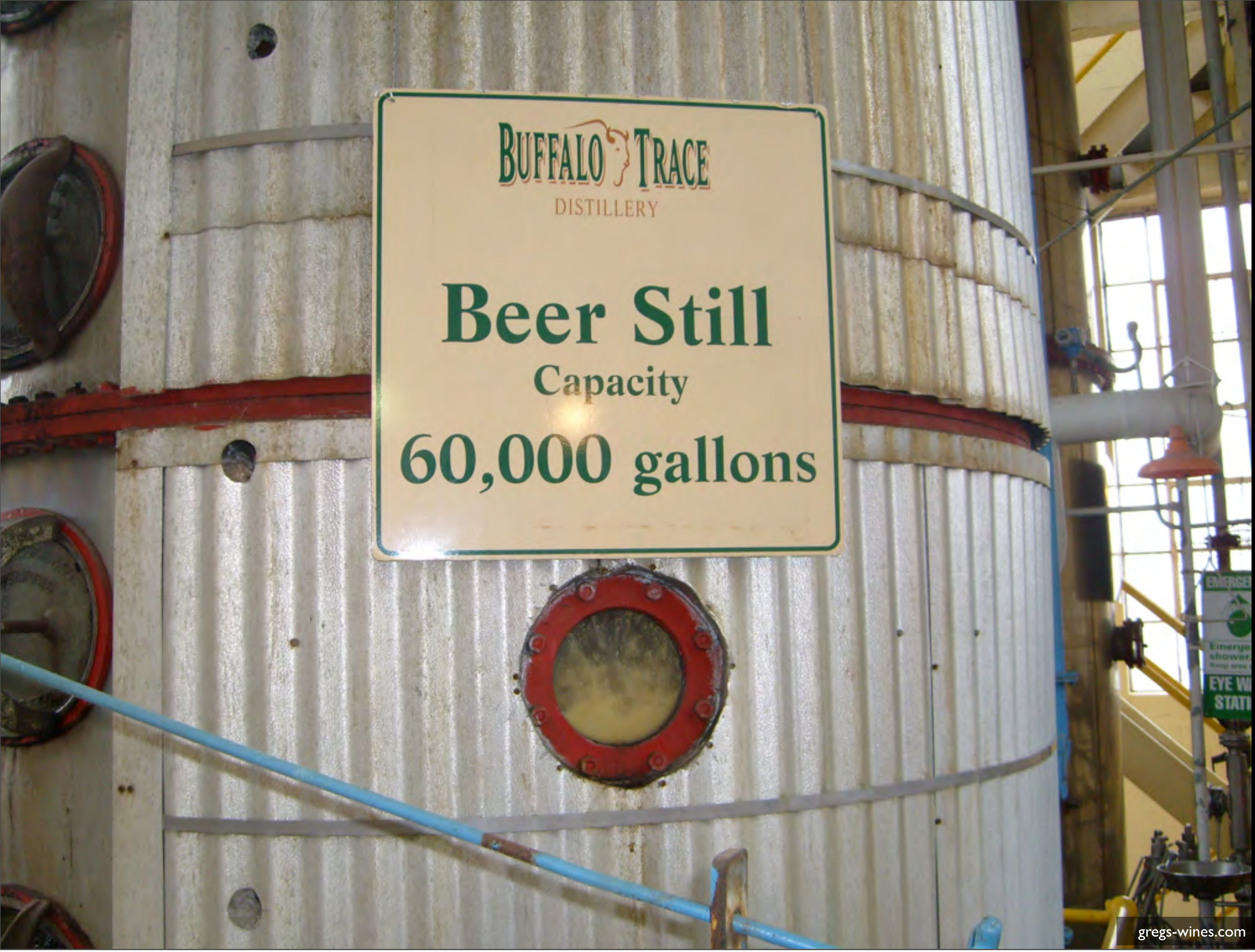
System V ABI Intel386 Architecture Processor Supplement (377 pages)

MIPS, ARM, PPC, and IA-64 too!

mac osx x86-64 calling convention

based on

System V ABI AMD64 Architecture Processor Supplement


BUFFALO TRACE
DISTILLERY
Beer Still
Capacity
60,000 gallons
EMERGE
EYE W
STATI

alignment

thomasgroup.com

end of argument area must be aligned on a 16byte boundary.

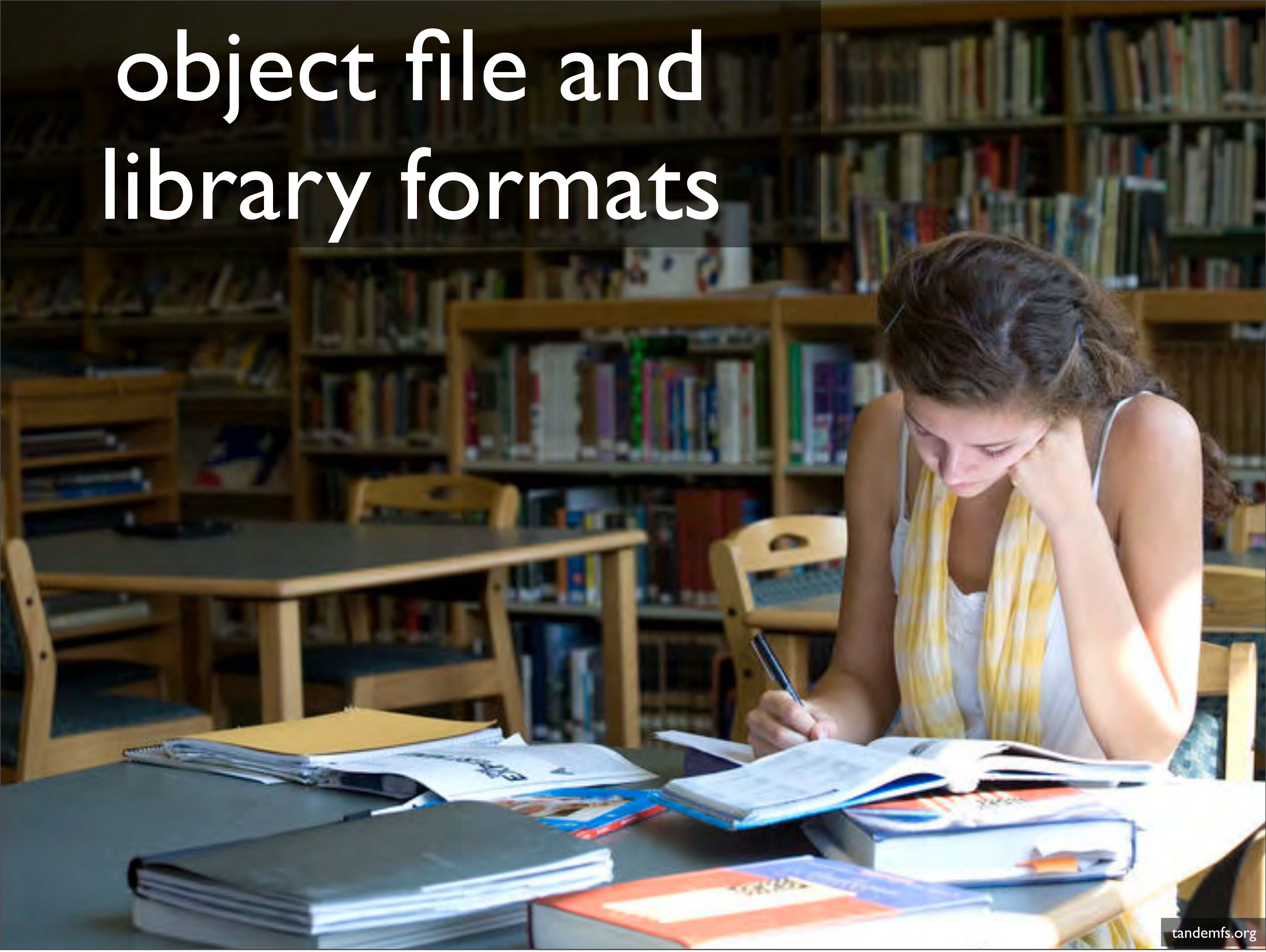
```
and $0xfffffffffffffff0, %rsp
```

calling convention

arianlim.wordpress.com

- function arguments from left to right live in: `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, `%r9`
- that’s for INTEGER class items.
- Other stuff gets passed on the stack (like on i386).
- registers are either caller or callee save

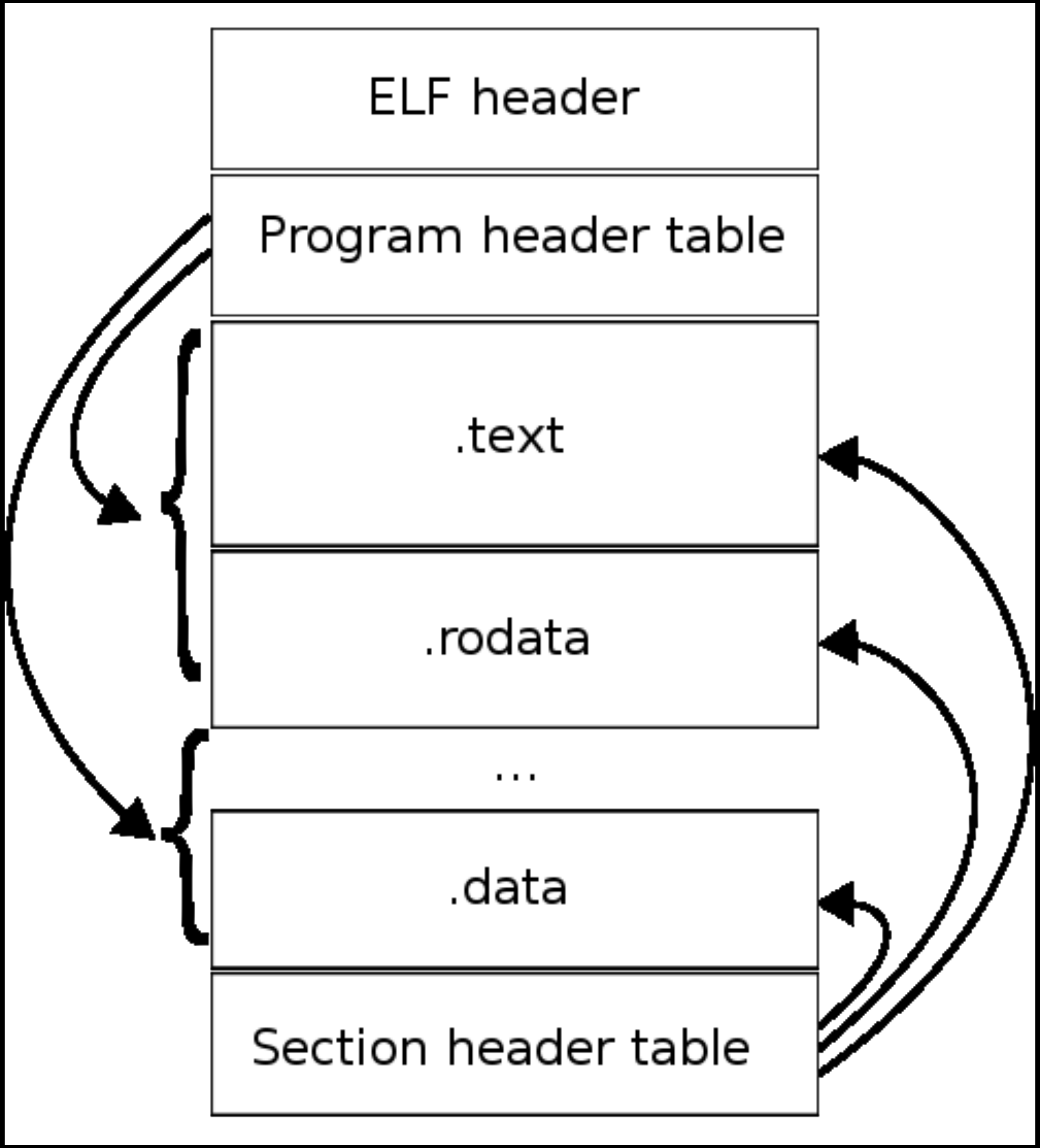
object file and library formats



tandemfs.org

steverubel.typepad.com

ELF Objects



en.wikipedia.org

ELF Objects

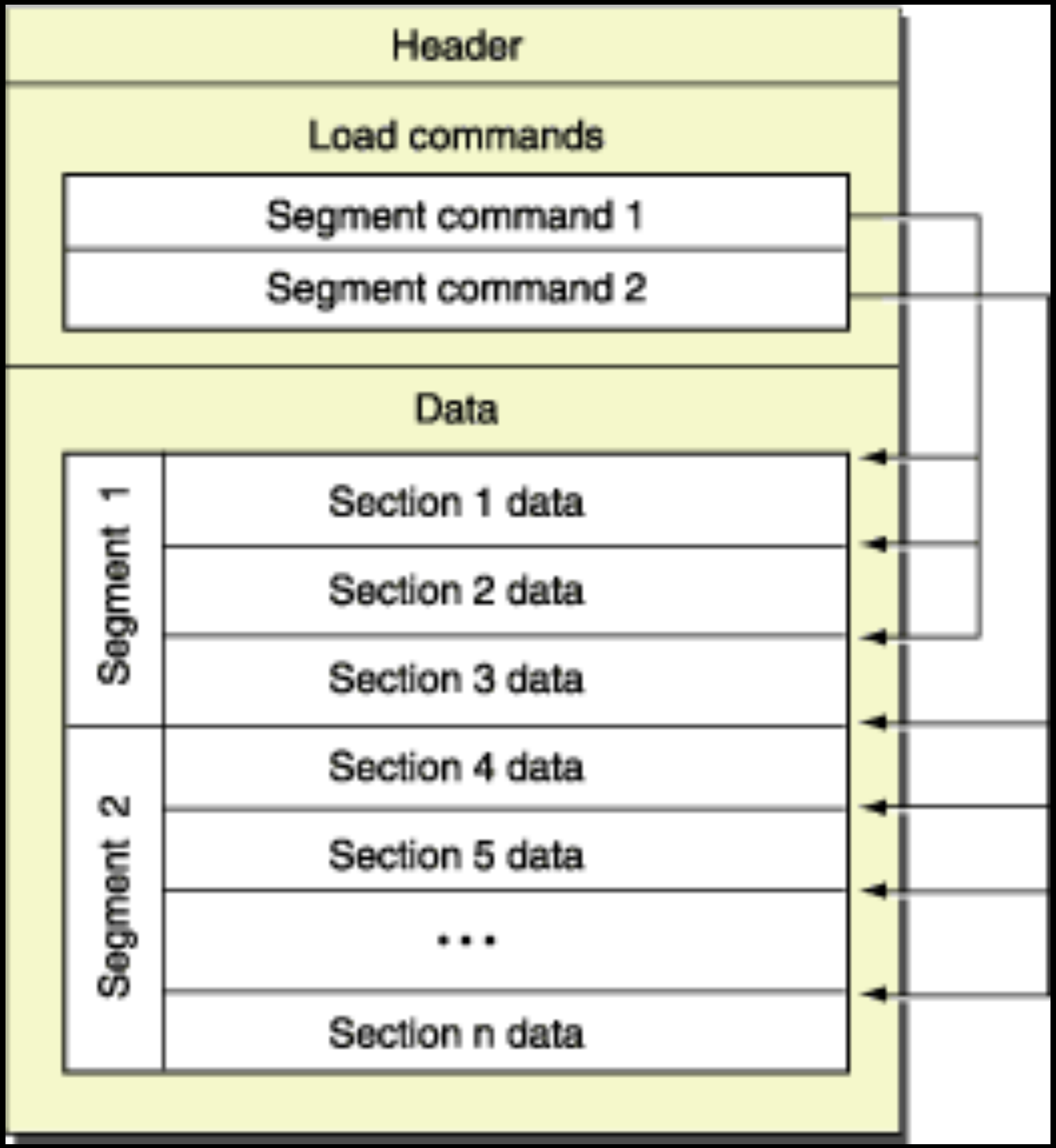
- ELF objects have headers
  - elf header (describes the elf object)
  - program headers (describes segments)
  - section headers (describes sections)
- libelf is useful for wandering the elf object extracting information.
- the executable and each .so has its own set of data

ELF Object sections

- .text - code lives here
- .plt - stub code that helps to “resolve” absolute function addresses.
- .got.plt - absolute function addresses; used by .plt entries.
- .debug_info - debugging information
- .gnu_debuglink - checksum and filename for debug info
- and more.

vanachteren.net

Mach-O Objects



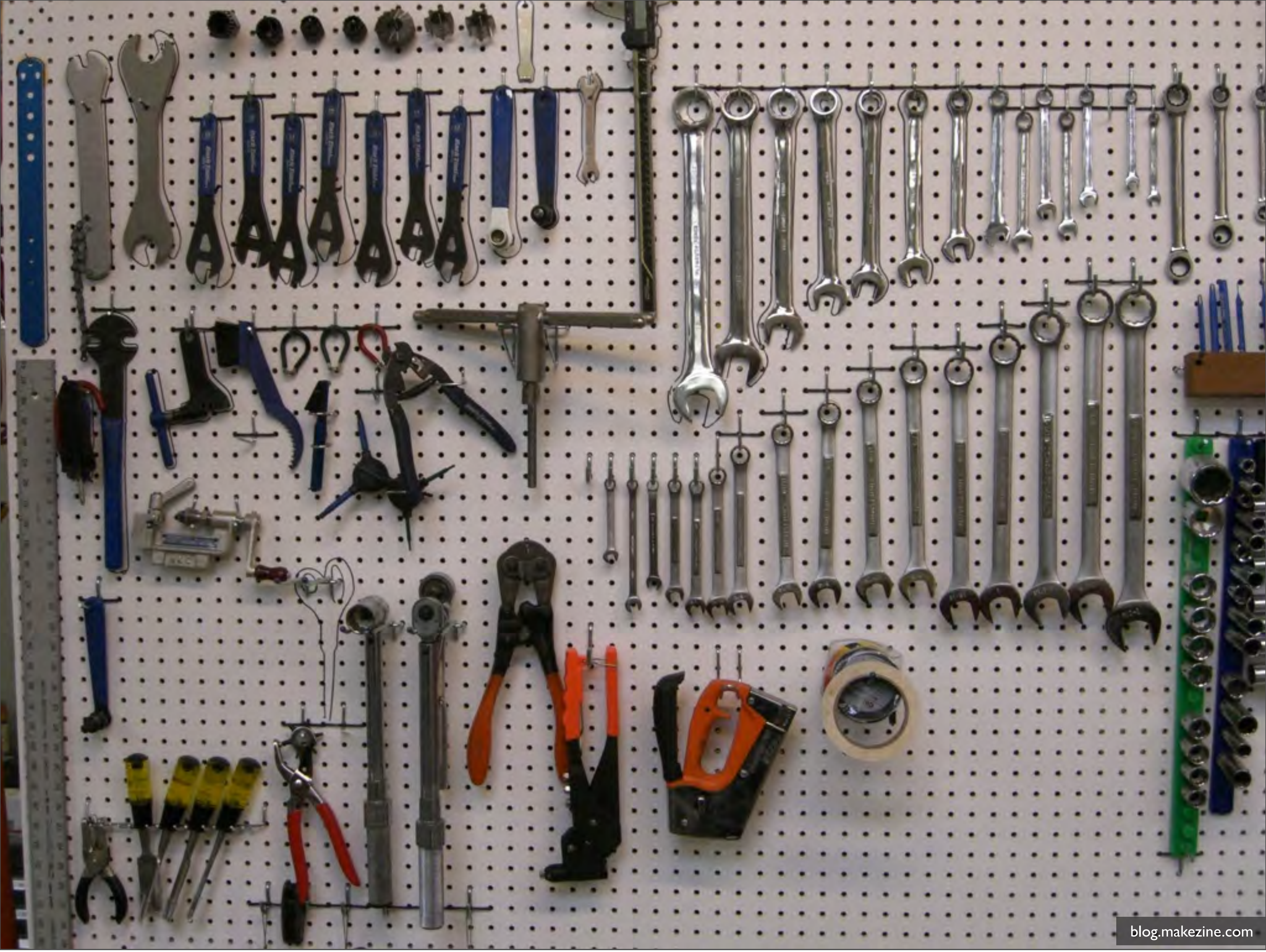
developer.apple.com

Mach-O Objects

- Mach-O objects have load commands
  - header (describes the mach-o object)
  - load commands (describe layout and linkage info)
  - segment commands (describes sections)
- dyld(3) describes some apis for touching mach-o objects
- <u>the executable and each dylib/bundle has its own set of data</u>

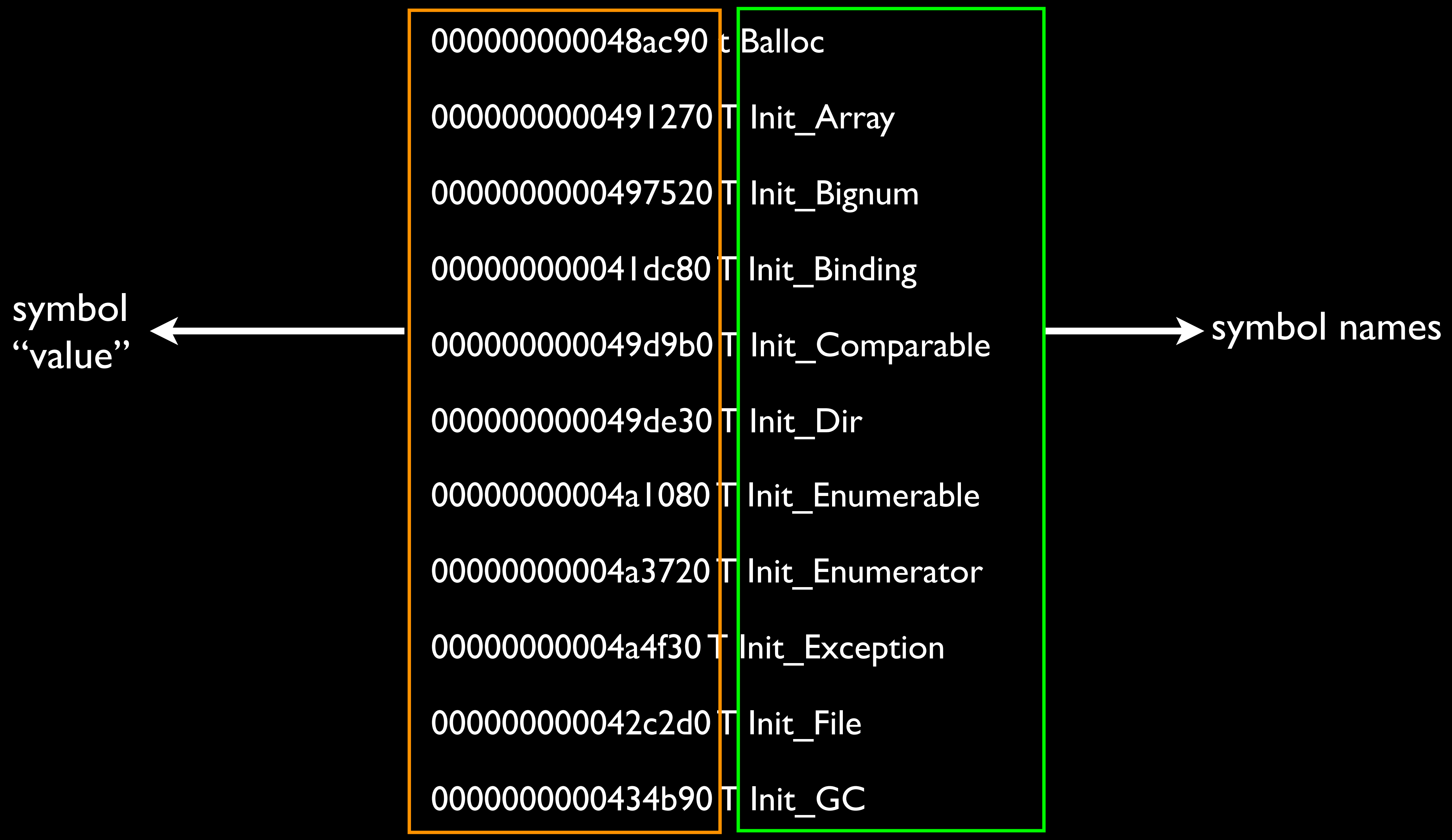
Mach-O sections

- __text - code lives here
- __symbol_stub1 - list of jmpq instructions for runtime dynamic linking
- __stub_helper - stub code that helps to “resolve” absolute function addresses.
- __la_symbol_ptr - absolute function addresses; used by symbol stub
- and more.


blog.makezine.com

nm

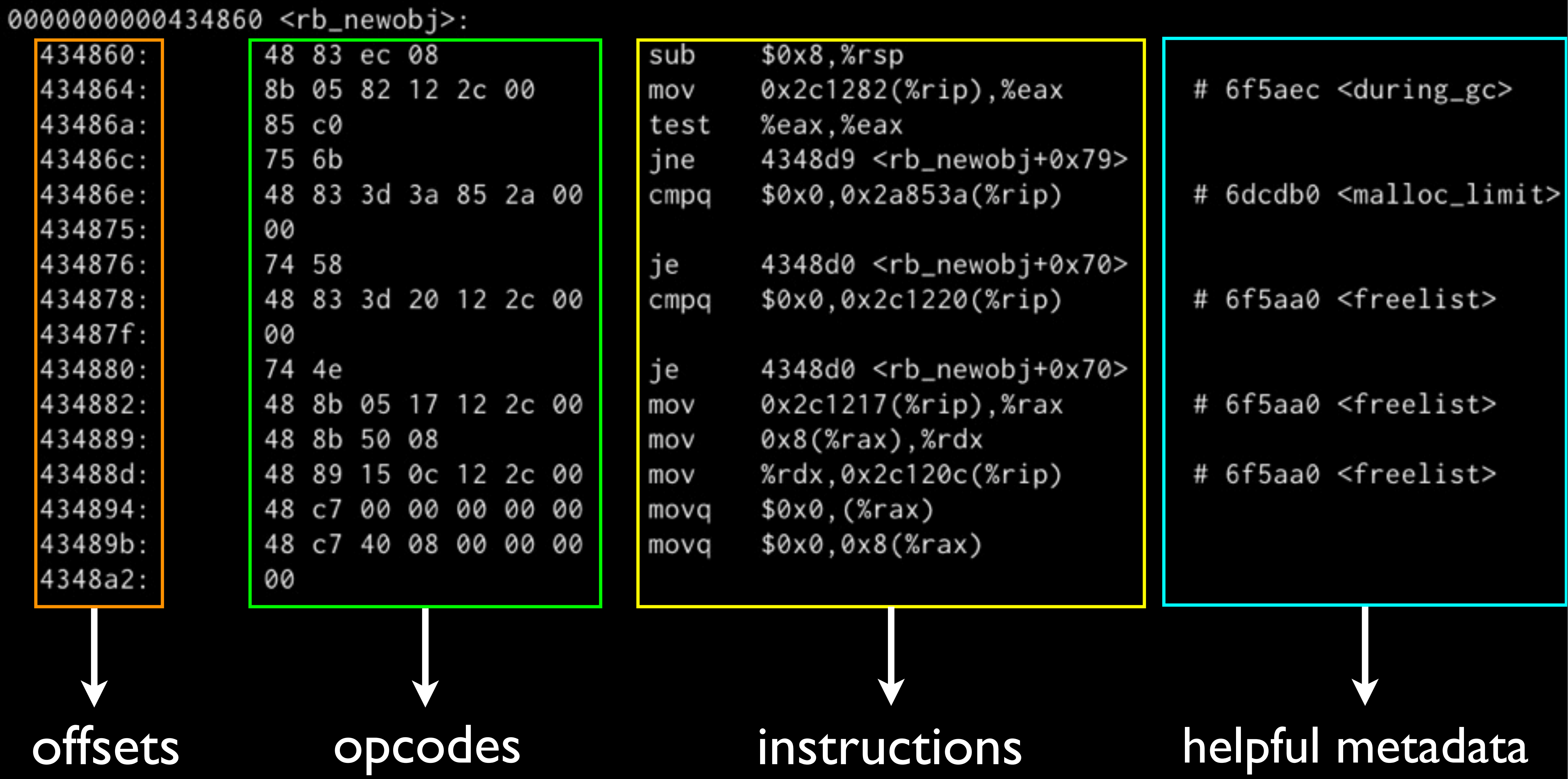
```
% nm /usr/bin/ruby
```



```
000000000048ac90 t Balloc
0000000000491270 T Init_Array
0000000000497520 T Init_Bignum
000000000041dc80 T Init_Binding
000000000049d9b0 T Init_Comparable
000000000049de30 T Init_Dir
00000000004a1080 T Init_Enumerable
00000000004a3720 T Init_Enumerator
00000000004a4f30 T Init_Exception
000000000042c2d0 T Init_File
0000000000434b90 T Init_GC
```

objdump

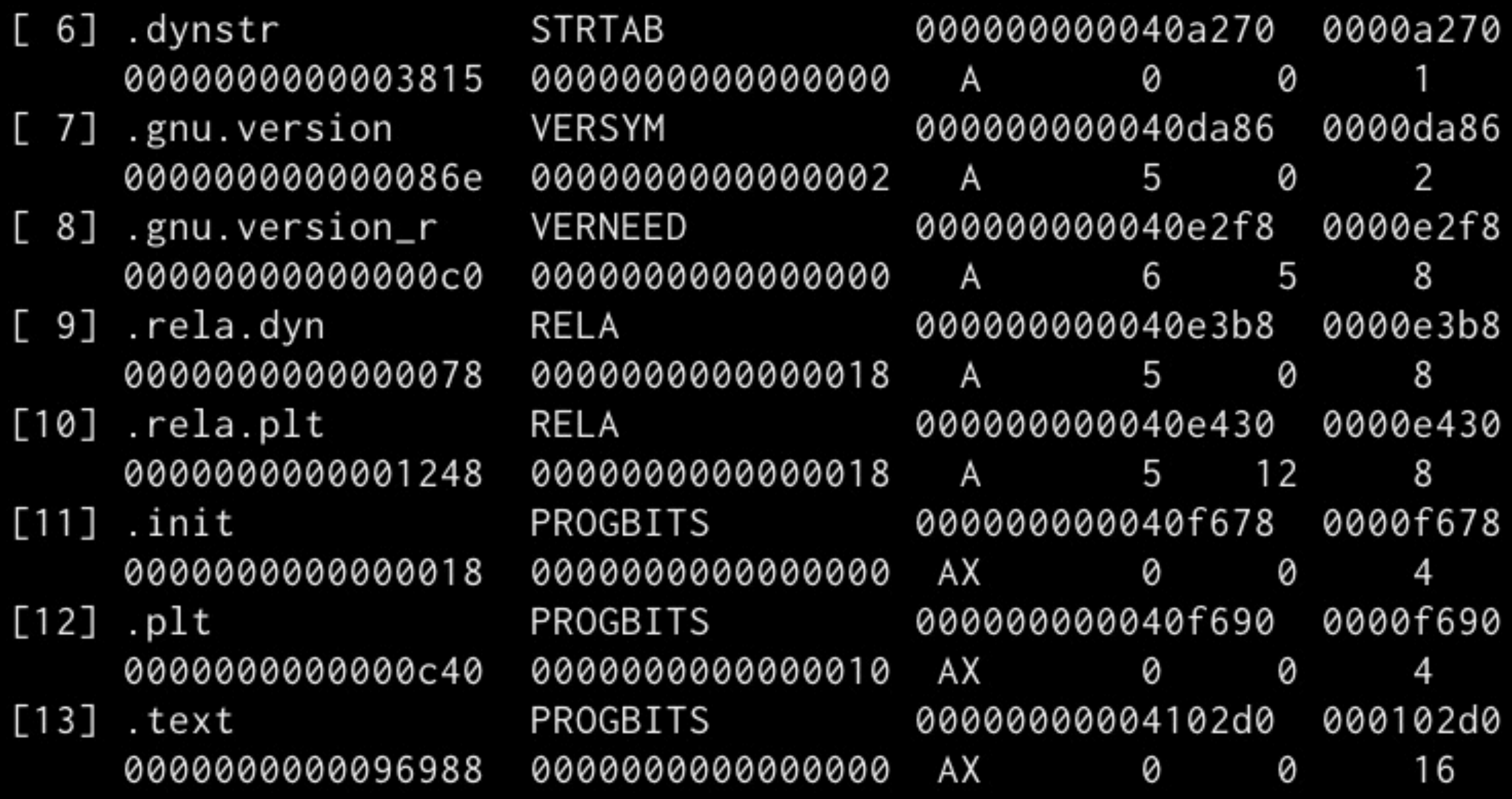
% objdump -D /usr/bin/ruby



```
0000000000434860 <rb_newobj>:
  434860:       48 83 ec 08             sub     $0x8,%rsp
  434864:       8b 05 82 12 2c 00       mov     0x2c1282(%rip),%eax       # 6f5aec <during_gc>
  43486a:       85 c0                   test    %eax,%eax
  43486c:       75 6b                   jne     4348d9 <rb_newobj+0x79>
  43486e:       48 83 3d 3a 85 2a 00    cmpq    $0x0,0x2a853a(%rip)       # 6dcdb0 <malloc_limit>
  434875:       00
  434876:       74 58                   je      4348d0 <rb_newobj+0x70>
  434878:       48 83 3d 20 12 2c 00    cmpq    $0x0,0x2c1220(%rip)       # 6f5aa0 <freelist>
  43487f:       00
  434880:       74 4e                   je      4348d0 <rb_newobj+0x70>
  434882:       48 8b 05 17 12 2c 00    mov     0x2c1217(%rip),%rax       # 6f5aa0 <freelist>
  434889:       48 8b 50 08             mov     0x8(%rax),%rdx
  43488d:       48 89 15 0c 12 2c 00    mov     %rdx,0x2c120c(%rip)       # 6f5aa0 <freelist>
  434894:       48 c7 00 00 00 00 00    movq    $0x0,(%rax)
  43489b:       48 c7 40 08 00 00 00    movq    $0x0,0x8(%rax)
  4348a2:       00
```

readelf

% readelf -a /usr/bin/ruby

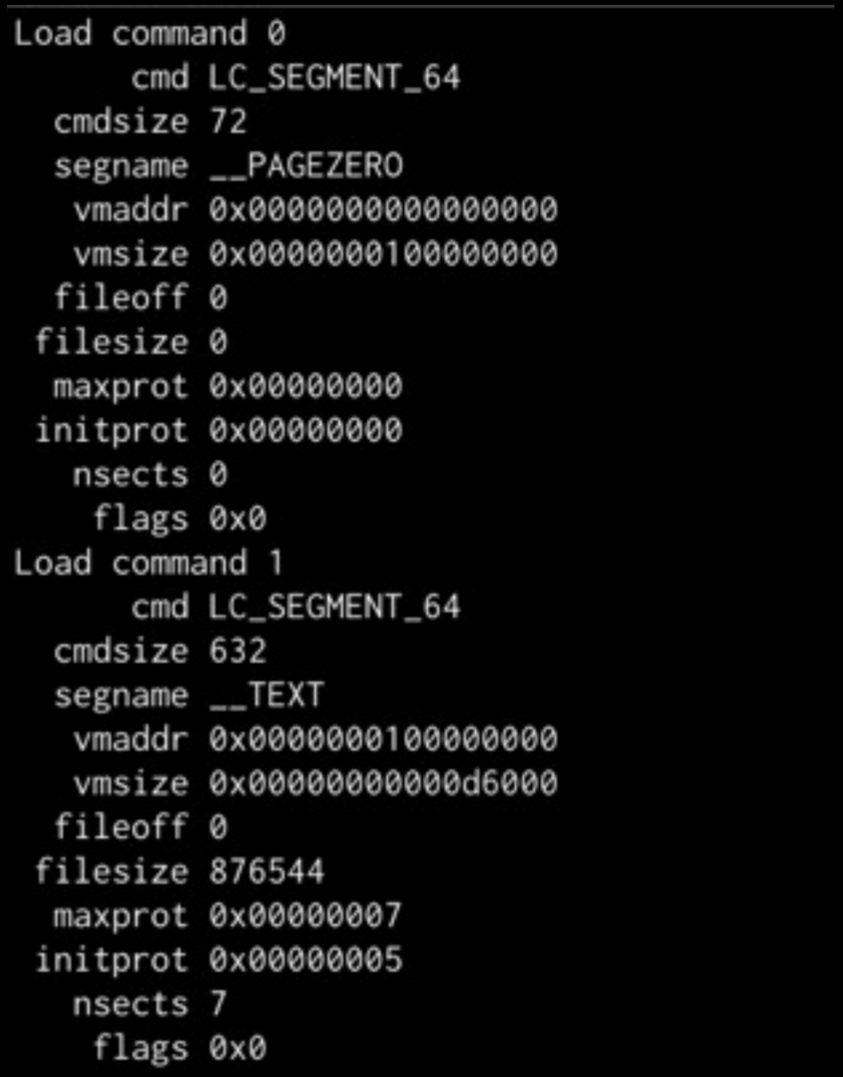


```
[ 6] .dynstr           STRTAB           000000000040a270  0000a270
     0000000000003815  0000000000000000   A       0     0     1
[ 7] .gnu.version      VERSYM           000000000040da86  0000da86
     000000000000086e  0000000000000002   A       5     0     2
[ 8] .gnu.version_r    VERNEED          000000000040e2f8  0000e2f8
     00000000000000c0  0000000000000000   A       6     5     8
[ 9] .rela.dyn         RELA             000000000040e3b8  0000e3b8
     0000000000000078  0000000000000018   A       5     0     8
[10] .rela.plt         RELA             000000000040e430  0000e430
     0000000000001248  0000000000000018   A       5    12     8
[11] .init             PROGBITS         000000000040f678  0000f678
     0000000000000018  0000000000000000  AX       0     0     4
[12] .plt              PROGBITS         000000000040f690  0000f690
     0000000000000c40  0000000000000010  AX       0     0     4
[13] .text             PROGBITS         00000000004102d0  000102d0
     0000000000096988  0000000000000000  AX       0     0     16
```

This is a *tiny* subset of the data available

otool

% otool -l /usr/bin/ruby



```
Load command 0
      cmd LC_SEGMENT_64
  cmdsize 72
  segname __PAGEZERO
   vmaddr 0x0000000000000000
   vmsize 0x0000000100000000
  fileoff 0
 filesize 0
  maxprot 0x00000000
 initprot 0x00000000
   nsects 0
    flags 0x0
Load command 1
      cmd LC_SEGMENT_64
  cmdsize 632
  segname __TEXT
   vmaddr 0x0000000100000000
   vmsize 0x00000000000d6000
  fileoff 0
 filesize 876544
  maxprot 0x00000007
 initprot 0x00000005
   nsects 7
    flags 0x0
```

This is a *tiny* subset of the data available

bassfishin.com

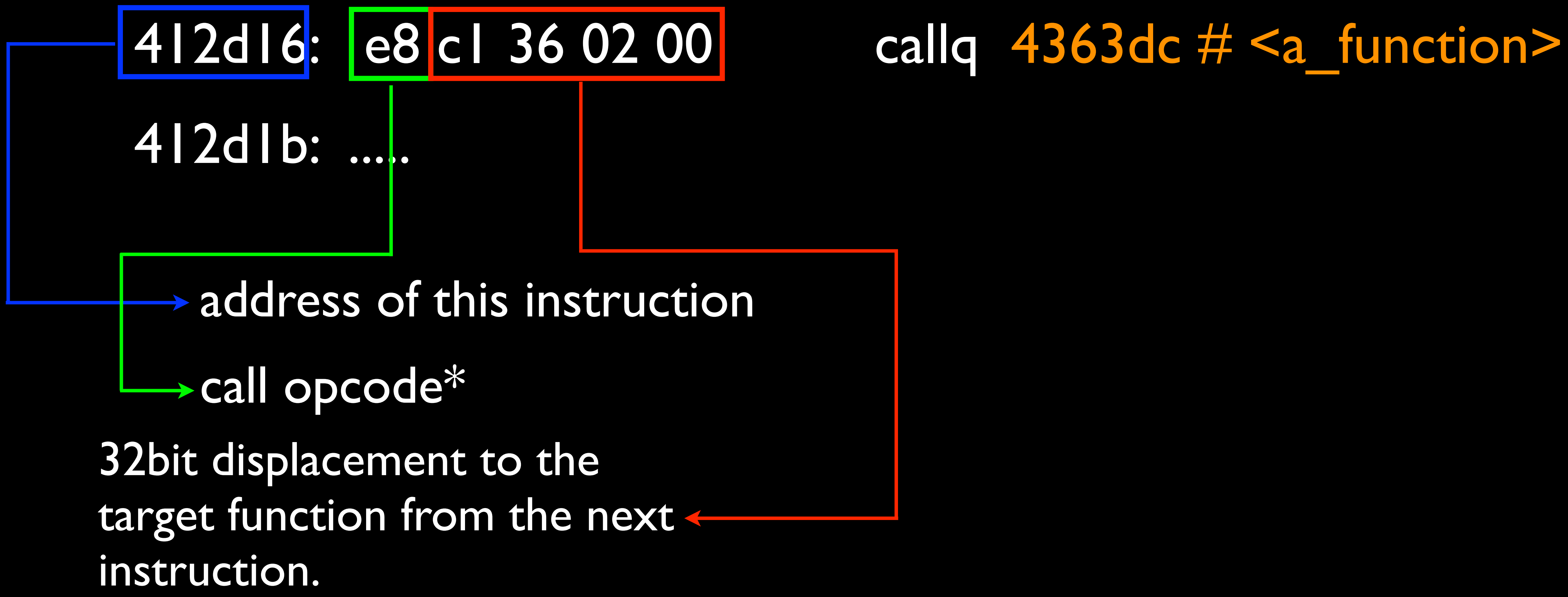
Calling functions

```
callq *%rbx

callq 0xdeadbeef
```

other ways, too...

anatomy of a call

(objdump output)



```
412d16:  e8 c1 36 02 00        callq  4363dc # <a_function>

412d1b:  .....
```

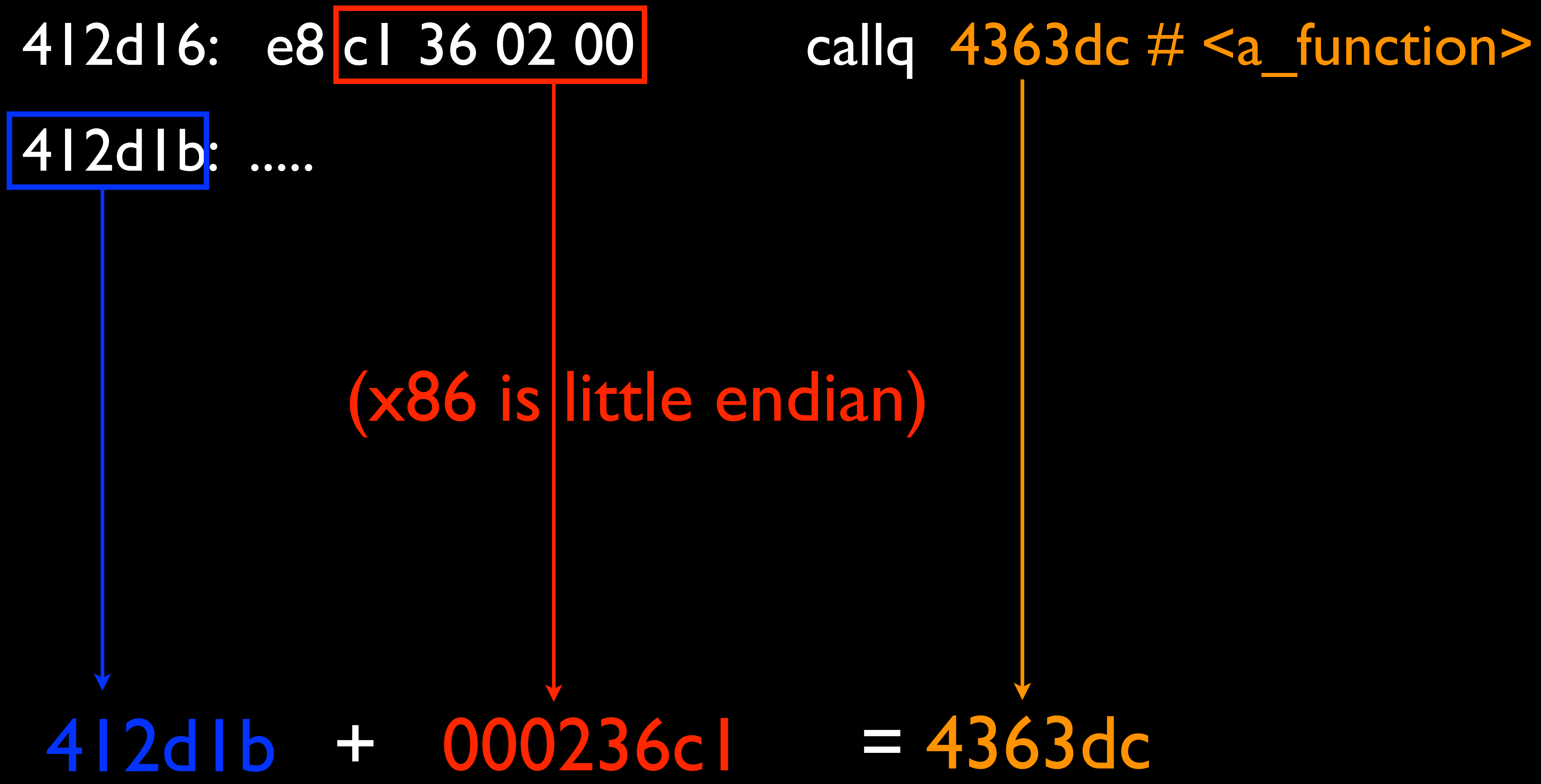
address of this instruction

call opcode*

32bit displacement to the target function from the next instruction.

anatomy of a call

(objdump output)



```
412d16:   e8 c1 36 02 00         callq  4363dc # <a_function>
412d1b: .....
```

(x86 is little endian)

412d1b + 000236c1 = 4363dc

Hook a_function

Overwrite the displacement so that all calls to a_function actually call a different function instead.

It may look like this:

```
int other_function()
{
        /* do something good/bad */

        /* be sure to call a_function! */
        return a_function();
}
```

codez are easy

```
/* CHILL, it’s fucking psuedo code */

while (are_moar_bytes()) {
  curr_ins = next_ins;
  next_ins = get_next_ins();
  if (curr_ins->type == INSN_CALL) {
    if ((hook_me - next_ins) == curr_ins->displacement) {
      /* found a call hook_me!*/
      rewrite(curr_ins->displacement, (replacement_fn - next_ins));
      return 0;
    }
  }
}
```

... right?.....

BE CAREFUL
NEAR MACHINERY

lemur.com

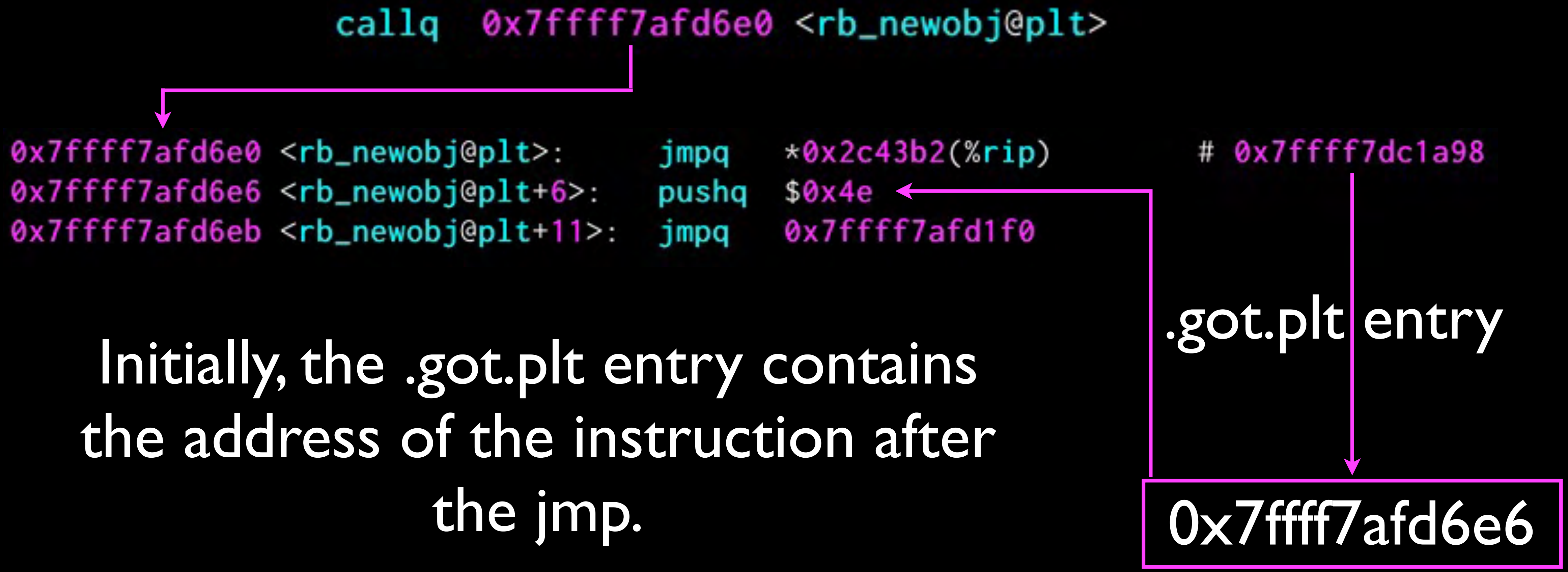
32bit displacement

- overwriting an existing call with another call
- stack will be aligned
- args are good to go
- can’t redirect to code that is outside of:
  - [rip+displacement]
- you can scan the address space looking for an available page with mmap, though...

Doesn’t work for all

- calling a function that is exported by a dynamic library **works differently.**

How runtime dynamic linking works (elf)

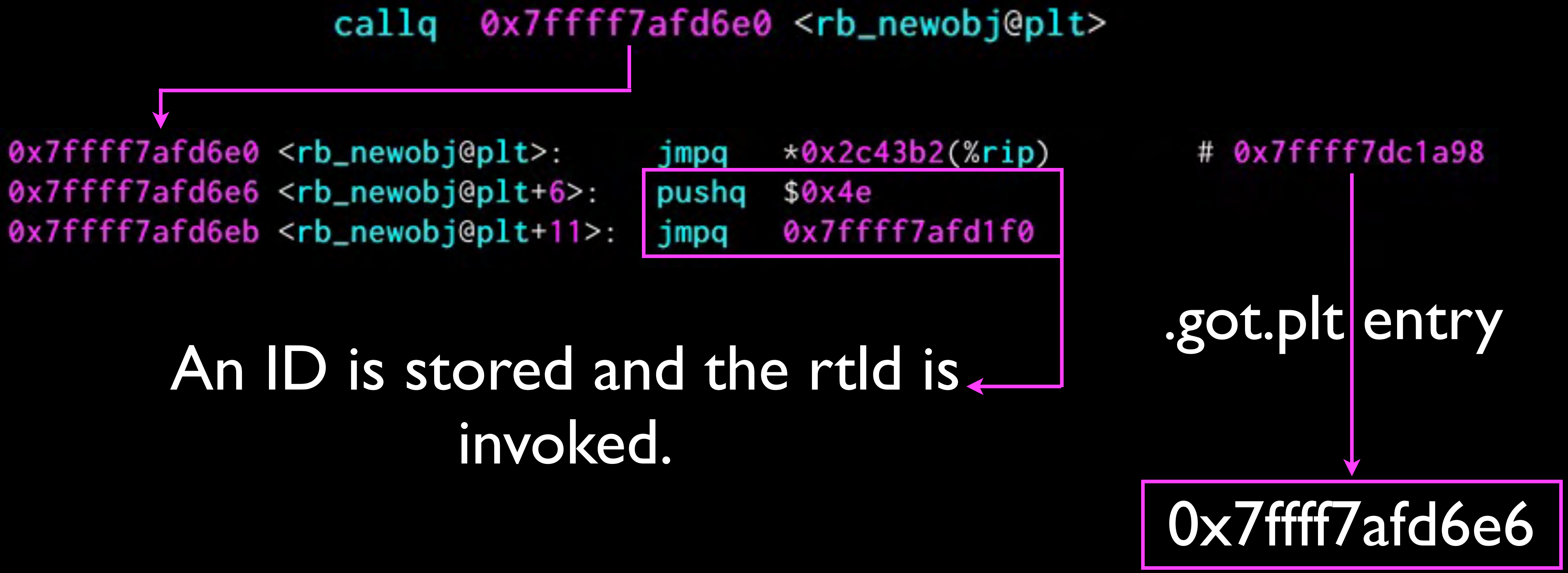


```
callq  0x7ffff7afd6e0 <rb_newobj@plt>
```

```
0x7ffff7afd6e0 <rb_newobj@plt>:     jmpq   *0x2c43b2(%rip)        # 0x7ffff7dc1a98
0x7ffff7afd6e6 <rb_newobj@plt+6>:   pushq  $0x4e
0x7ffff7afd6eb <rb_newobj@plt+11>:  jmpq   0x7ffff7afd1f0
```

Initially, the .got.plt entry contains the address of the instruction after the jmp.

How runtime dynamic linking works (elf)



```
callq  0x7ffff7afd6e0 <rb_newobj@plt>
```

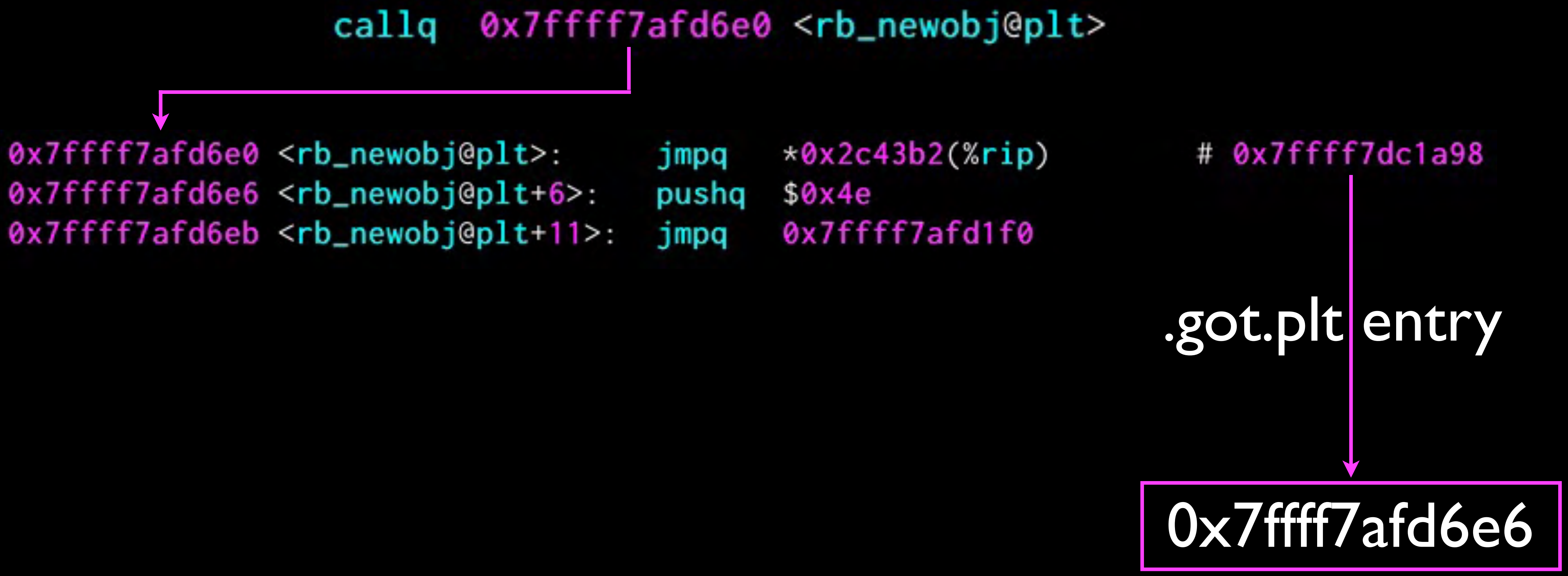
```
0x7ffff7afd6e0 <rb_newobj@plt>:     jmpq   *0x2c43b2(%rip)        # 0x7ffff7dc1a98
0x7ffff7afd6e6 <rb_newobj@plt+6>:   pushq  $0x4e
0x7ffff7afd6eb <rb_newobj@plt+11>:  jmpq   0x7ffff7afd1f0
```

An ID is stored and the rtld is invoked.

.got.plt entry

0x7ffff7afd6e6

How runtime dynamic linking works (elf)



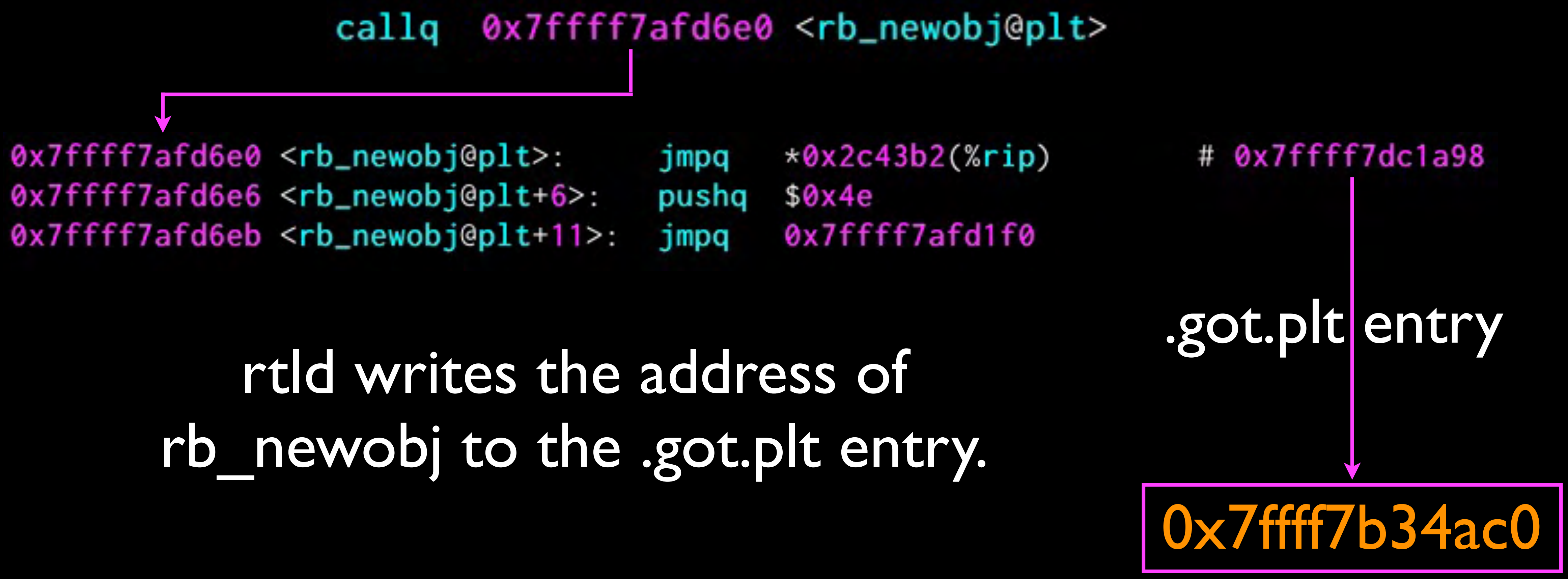
```
callq  0x7ffff7afd6e0 <rb_newobj@plt>
```

```
0x7ffff7afd6e0 <rb_newobj@plt>:     jmpq   *0x2c43b2(%rip)        # 0x7ffff7dc1a98
0x7ffff7afd6e6 <rb_newobj@plt+6>:   pushq  $0x4e
0x7ffff7afd6eb <rb_newobj@plt+11>:  jmpq   0x7ffff7afd1f0
```

.got.plt entry

0x7ffff7afd6e6

How runtime dynamic linking works (elf)



```
callq  0x7ffff7afd6e0 <rb_newobj@plt>
```

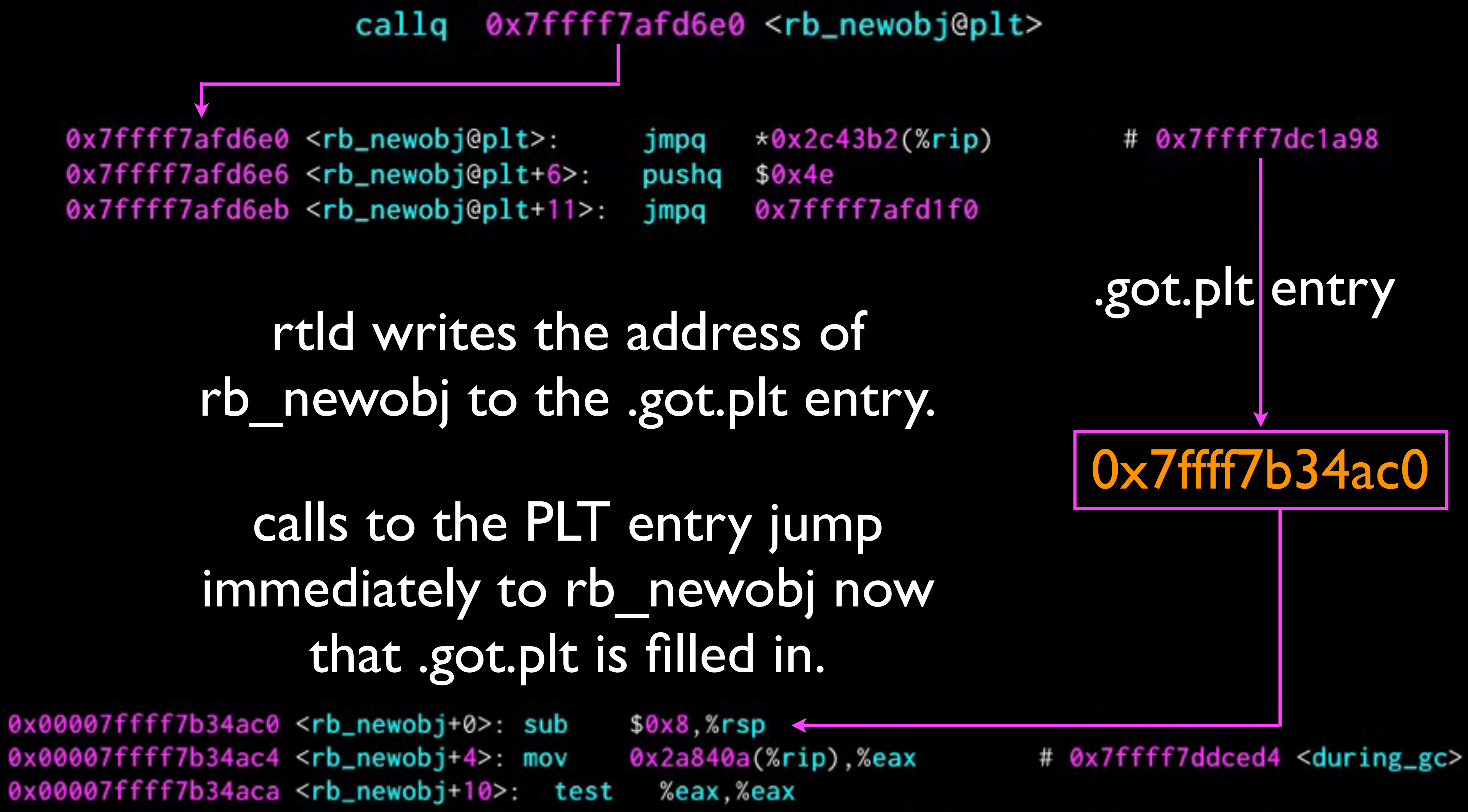
```
0x7ffff7afd6e0 <rb_newobj@plt>:     jmpq   *0x2c43b2(%rip)        # 0x7ffff7dc1a98
0x7ffff7afd6e6 <rb_newobj@plt+6>:   pushq  $0x4e
0x7ffff7afd6eb <rb_newobj@plt+11>:  jmpq   0x7ffff7afd1f0
```

.got.plt entry

rtld writes the address of rb_newobj to the .got.plt entry.

0x7ffff7b34ac0

How runtime dynamic linking works (elf)



```
callq  0x7ffff7afd6e0 <rb_newobj@plt>
```

```
0x7ffff7afd6e0 <rb_newobj@plt>:     jmpq   *0x2c43b2(%rip)        # 0x7ffff7dc1a98
0x7ffff7afd6e6 <rb_newobj@plt+6>:   pushq  $0x4e
0x7ffff7afd6eb <rb_newobj@plt+11>:  jmpq   0x7ffff7afd1f0
```

.got.plt entry

0x7ffff7b34ac0

rtld writes the address of rb_newobj to the .got.plt entry.

calls to the PLT entry jump immediately to rb_newobj now that .got.plt is filled in.

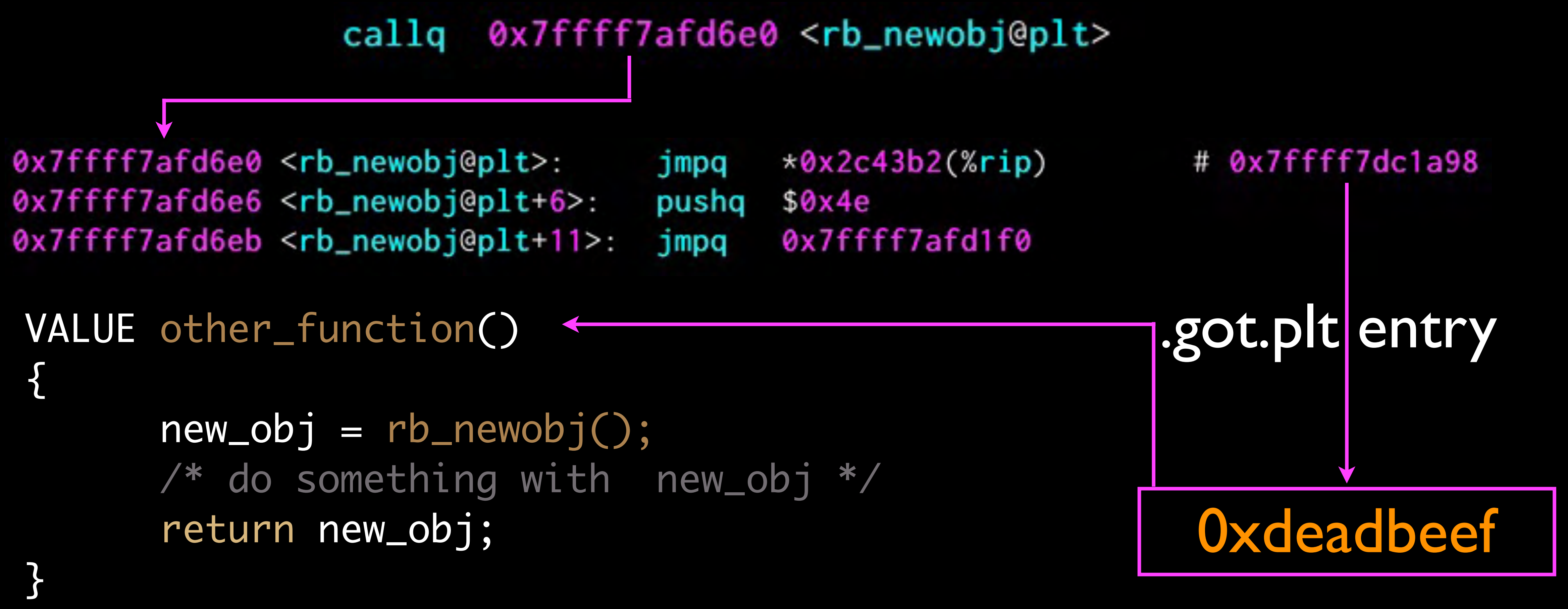
```
0x00007ffff7b34ac0 <rb_newobj+0>: sub    $0x8,%rsp
0x00007ffff7b34ac4 <rb_newobj+4>: mov    0x2a840a(%rip),%eax        # 0x7ffff7ddced4 <during_gc>
0x00007ffff7b34aca <rb_newobj+10>:  test   %eax,%eax
```

rs.tacklewarehouse.com

Hook the GOT

Redirect execution by overwriting all the .got.plt entries for rb_newobj in each DSO with a handler function instead.

Hook the GOT



```
callq  0x7ffff7afd6e0 <rb_newobj@plt>
```

```
0x7ffff7afd6e0 <rb_newobj@plt>:     jmpq   *0x2c43b2(%rip)        # 0x7ffff7dc1a98
0x7ffff7afd6e6 <rb_newobj@plt+6>:   pushq  $0x4e
0x7ffff7afd6eb <rb_newobj@plt+11>:  jmpq   0x7ffff7afd1f0
```

```
VALUE other_function()
{
      new_obj = rb_newobj();
      /* do something with  new_obj */
      return new_obj;
}
```

.got.plt entry

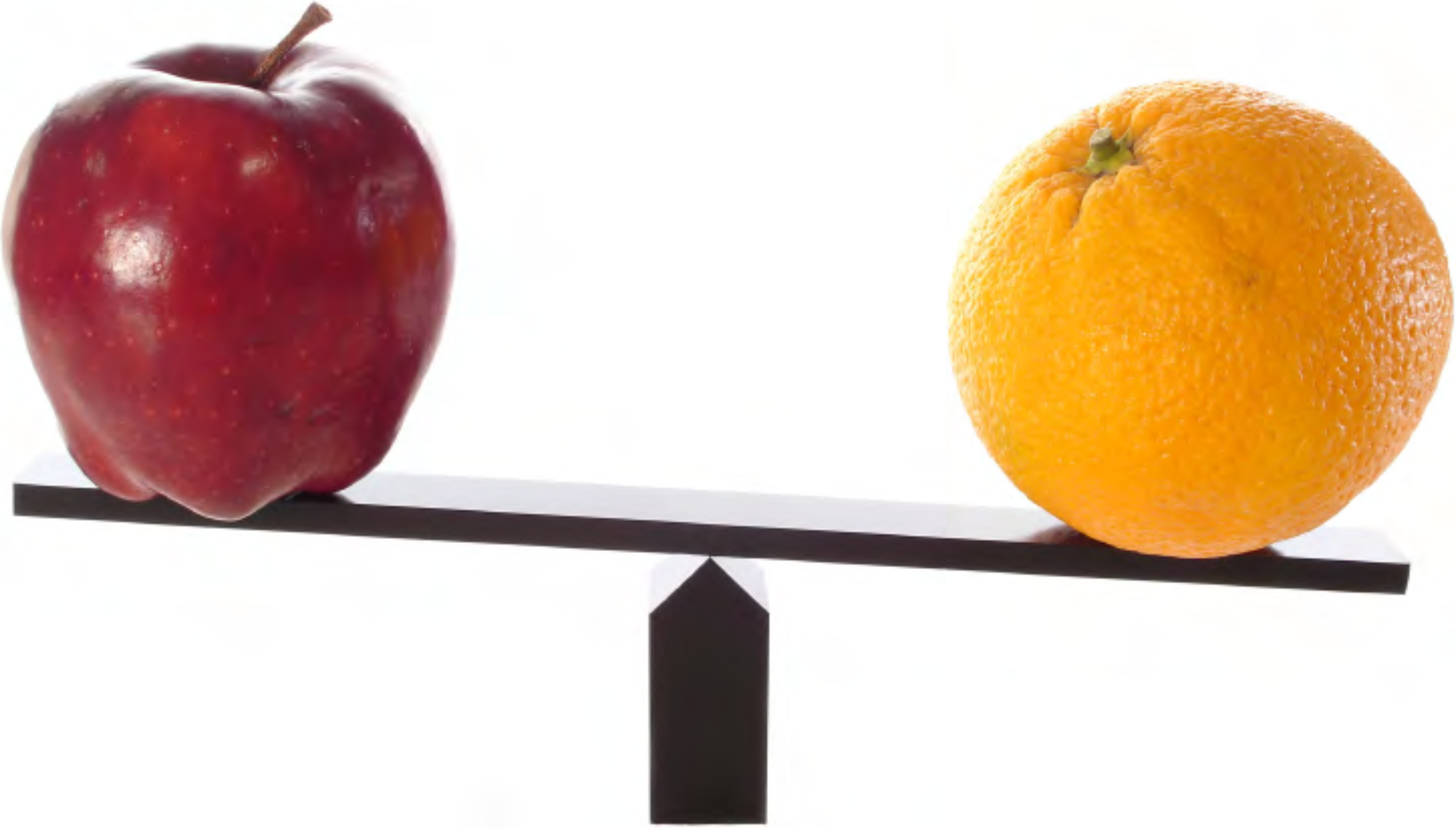
0xdeadbeef

WAIT... other_function() calls rb_newobj() isn’t that an infinite loop?

NO, it isn’t. other_function() lives in it’s own DSO, so its calls to rb_newobj() use the .plt/.got.plt in its own DSO.

As long as we leave other_function()‘s DSO unmodified, we’ll avoid an infinite loop.

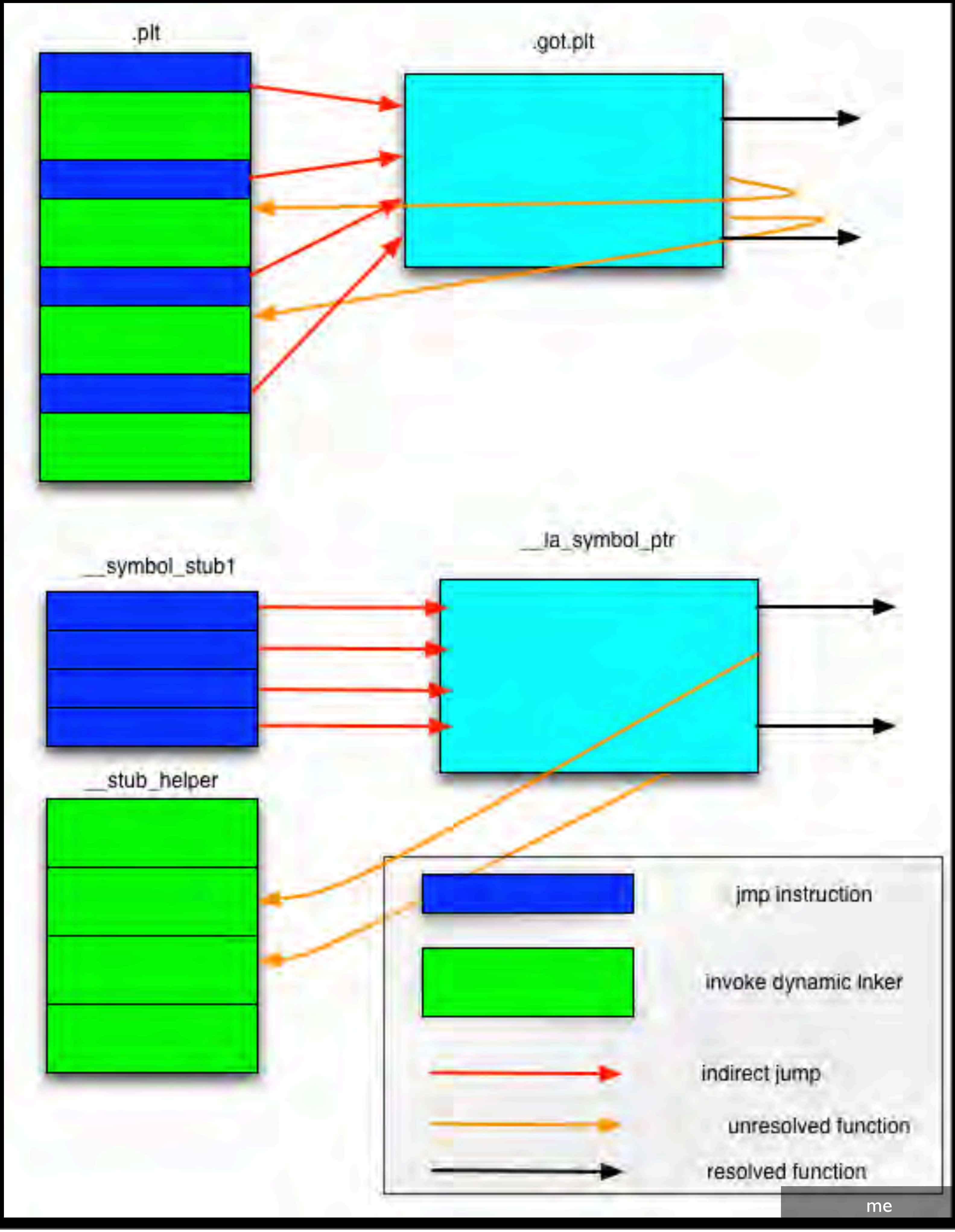
vanachteren.net



tlaneve.files.wordpress.com

elf

mach-o


.plt
.got.plt
__symbol_stub1
__la_symbol_ptr
__stub_helper
jmp instruction
invoke dynamic lnker
indirect jump
unresolved function
resolved function
me

what else is left?

inline functions.

add_freelist

- Can’t hook because add_freelist is *inlined*:

```
static inline void
add_freelist(p)
    RVALUE *p
{
    p->as.free.flags = 0;
    p->as.free.next = freelist;
    freelist = p;
}
```

- The compiler has the option of inserting the instructions of this function directly into the callers.
- If this happens, you won’t see any calls.

So... what now?

- Look carefully at the generated code:

```
static inline void
add_freelist(p)
    RVALUE *p
{
    p->as.free.flags = 0;
    p->as.free.next = freelist;
    freelist = p;
}
```

- Notice that freelist gets updated.
- freelist has file level scope.
- hmmmm.....

A (stupid) crazy idea

- freelist has file level scope and lives at some static address.
- add_freelist updates freelist, so...
- Why not *search the binary for mov instructions that have freelist as the target!*
- Overwrite that mov instruction with a call to our code!
- But... we have a problem.
- The system isn’t ready for a call instruction.

alignment

thomasgroup.com

calling convention

arianlim.wordpress.com

Isn't ready? What?

- The 64bit ABI says that the stack must be aligned to a 16byte boundary after any/all arguments have been arranged.
- Since the overwrite is just some random mov, no way to guarantee that the stack is aligned.
- If we just plop in a call instruction, we won't be able to arrange for arguments to get put in the right registers.
- So now what?

jmp

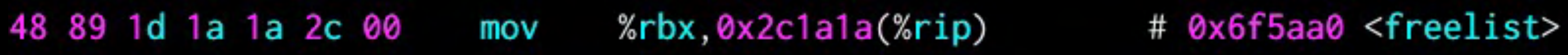
- Can use a jmp instruction.
- Transfer execution to an assembly stub **generated at runtime.**
  - recreate the overwritten instruction
  - set the system up to call a function
- do something good/bad
- jmp back when done to resume execution

picasaweb.google.com/lh/photo/-R3BPlqOq8MfQGFTduIqCA

checklist

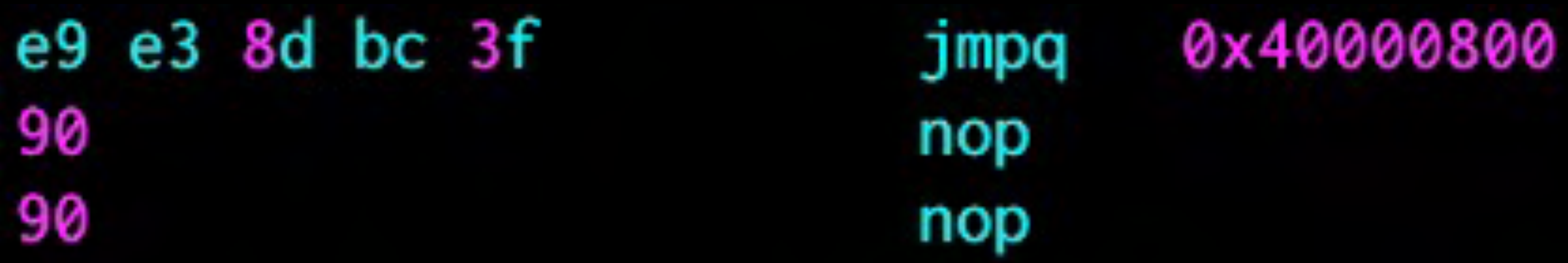
- save and restore caller/callee saved registers.
- align the stack.
- recreate what was overwritten.
- arrange for any arguments your replacement function needs to end up in registers.
- invoke your code.
- resume execution as if nothing happened.

this instruction updates the freelist and comes from add_freelist:



```
48 89 1d 1a 1a 2c 00    mov    %rbx,0x2c1a1a(%rip)        # 0x6f5aa0 <freelist>
```

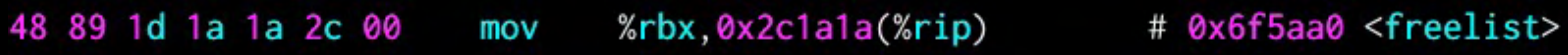
Can’t overwrite it with a call instruction because the state of the system is not ready for a function call.



```
e9 e3 8d bc 3f          jmpq   0x40000800
90                      nop
90                      nop
```

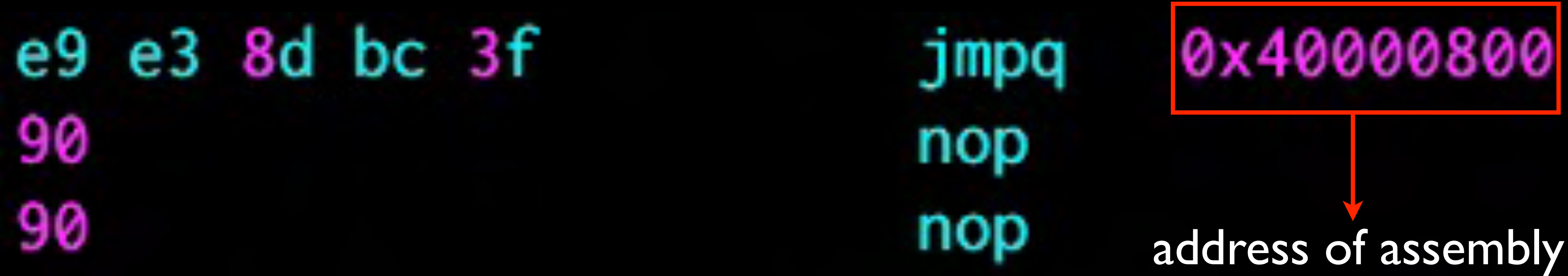
The jmp instruction and its offset are 5 bytes wide. Can’t grow or shrink the binary, so insert 2 one byte NOPs.

this instruction updates the freelist and comes from add_freelist:



```
48 89 1d 1a 1a 2c 00    mov    %rbx,0x2c1a1a(%rip)        # 0x6f5aa0 <freelist>
```

Can’t overwrite it with a call instruction because the state of the system is not ready for a function call.

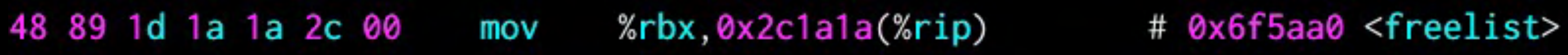


```
e9 e3 8d bc 3f          jmpq   0x40000800
90                      nop
90                      nop
```

address of assembly stub

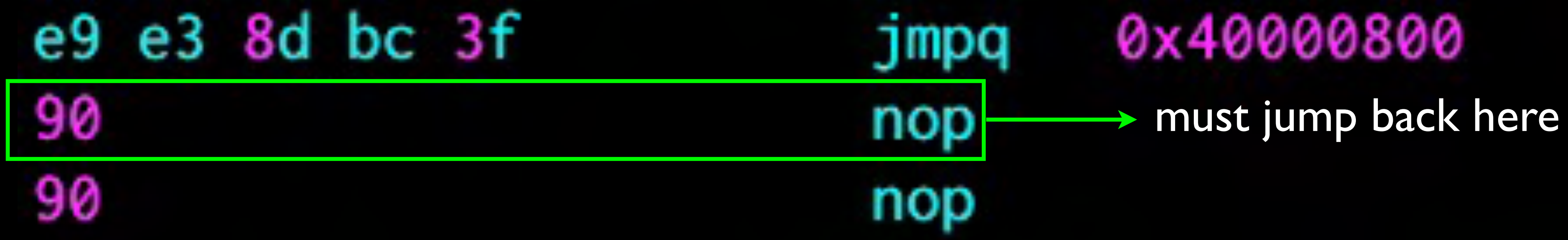
The jmp instruction and its offset are 5 bytes wide. Can’t grow or shrink the binary, so insert 2 one byte NOPs.

this instruction updates the freelist and comes from add_freelist:



```
48 89 1d 1a 1a 2c 00    mov    %rbx,0x2c1a1a(%rip)        # 0x6f5aa0 <freelist>
```

Can’t overwrite it with a call instruction because the state of the system is not ready for a function call.

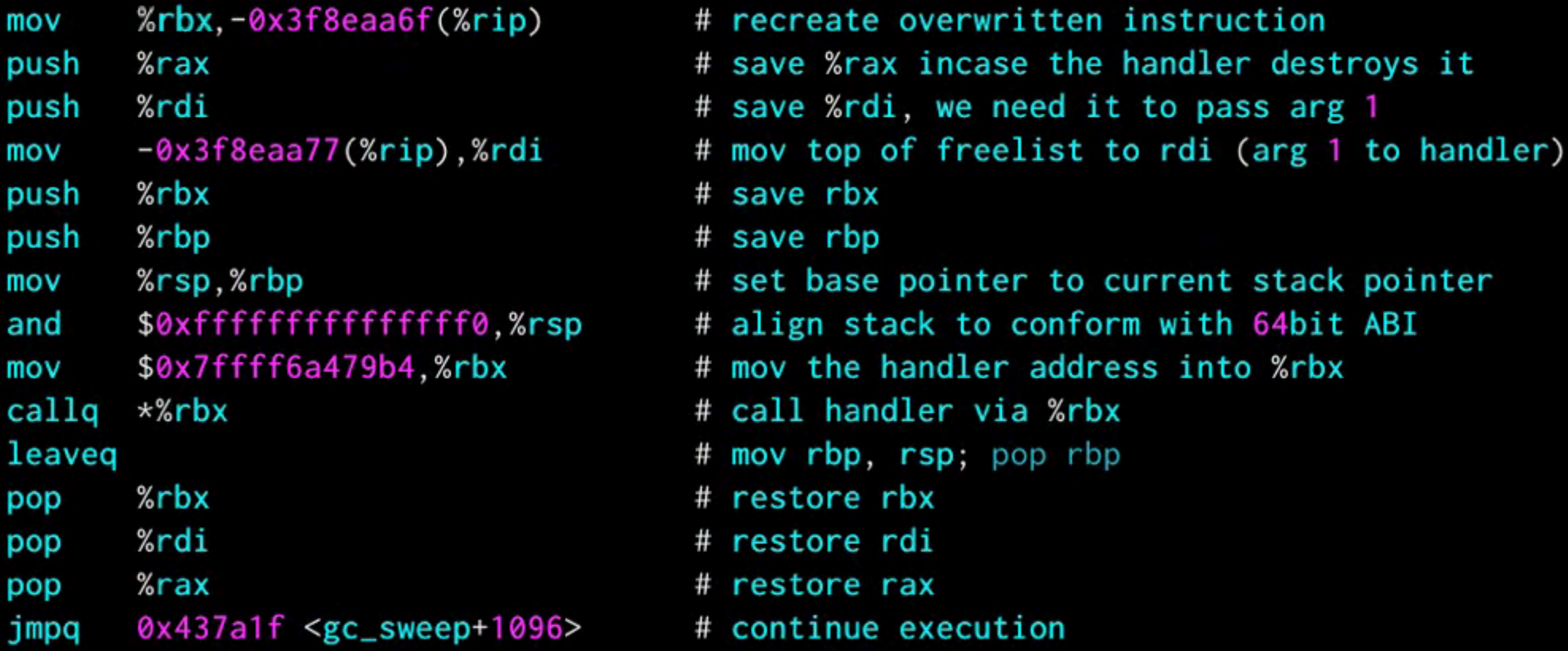


```
e9 e3 8d bc 3f          jmpq   0x40000800
90                      nop
90                      nop
```

must jump back here

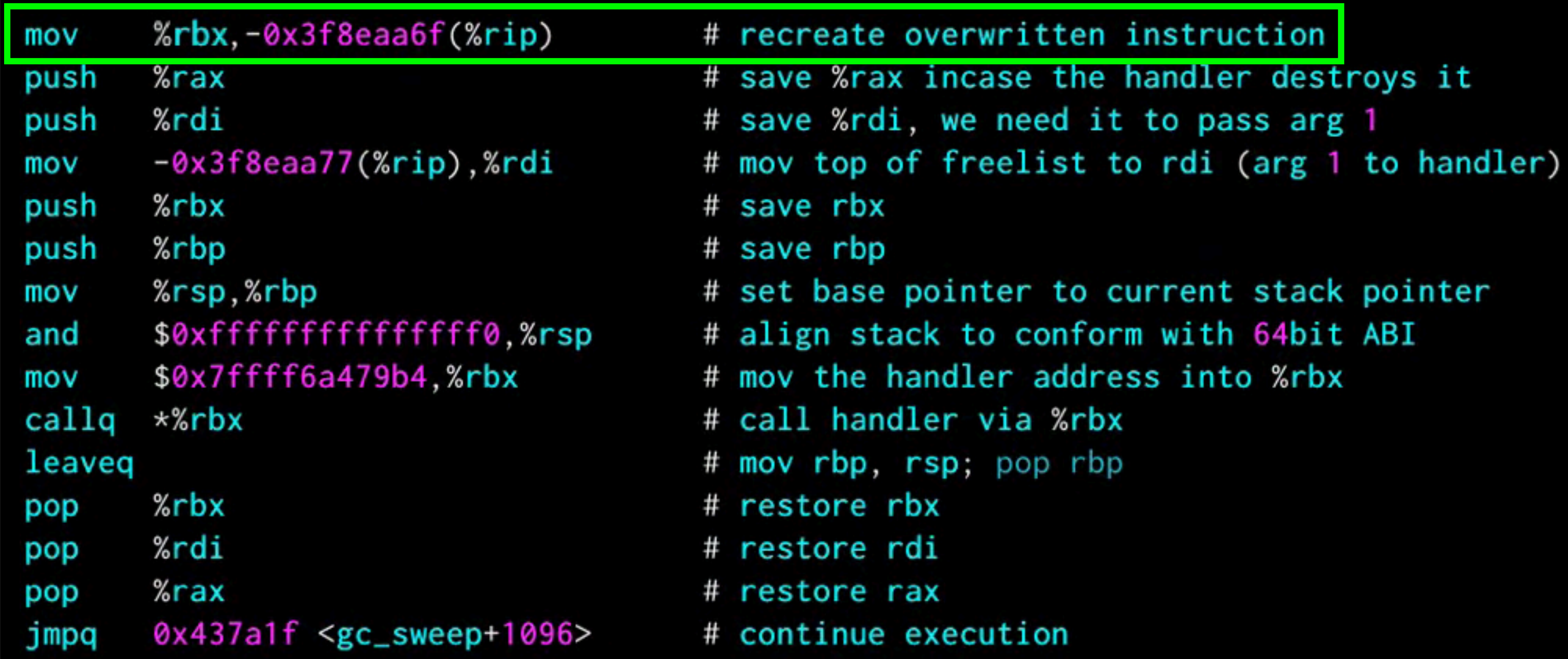
The jmp instruction and its offset are 5 bytes wide. Can’t grow or shrink the binary, so insert 2 one byte NOPs.

shortened assembly stub



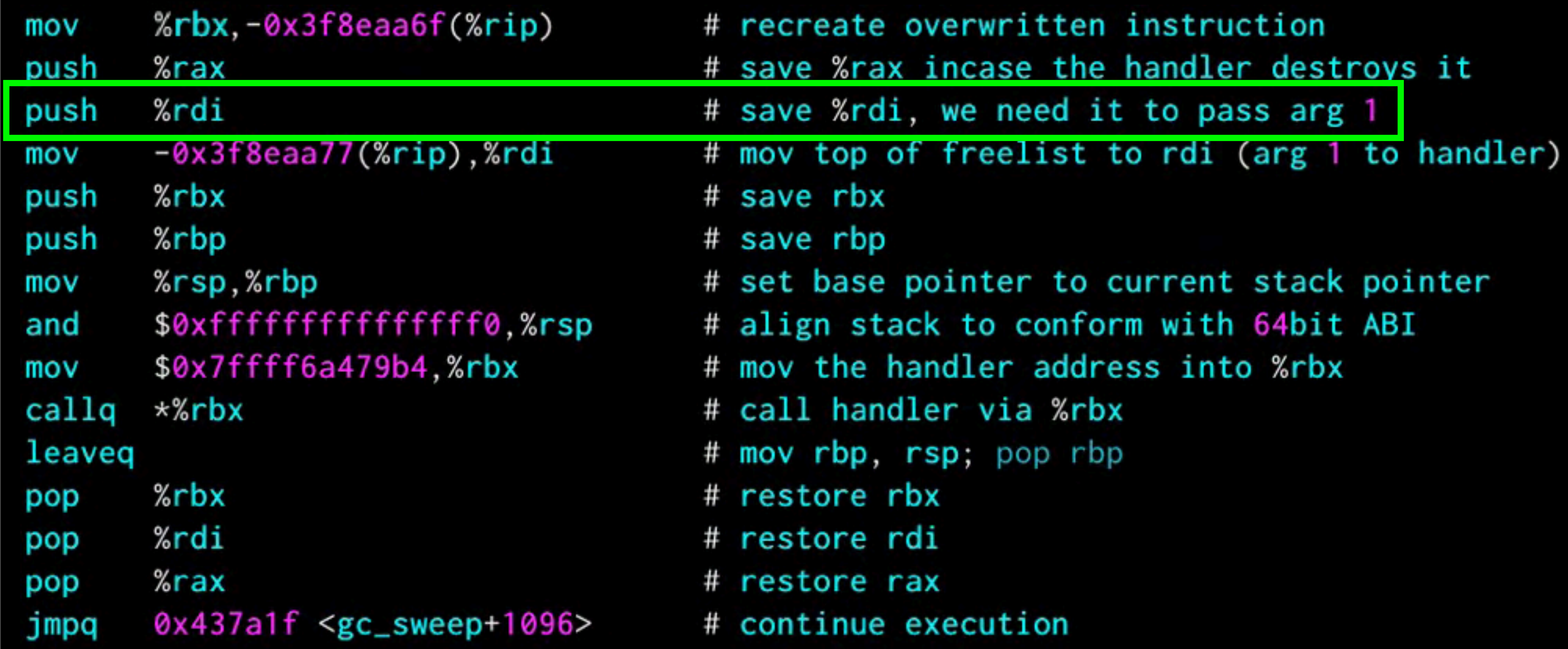
```
mov     %rbx,-0x3f8eaa6f(%rip)          # recreate overwritten instruction
push    %rax                            # save %rax incase the handler destroys it
push    %rdi                            # save %rdi, we need it to pass arg 1
mov     -0x3f8eaa77(%rip),%rdi          # mov top of freelist to rdi (arg 1 to handler)
push    %rbx                            # save rbx
push    %rbp                            # save rbp
mov     %rsp,%rbp                       # set base pointer to current stack pointer
and     $0xfffffffffffffff0,%rsp        # align stack to conform with 64bit ABI
mov     $0x7ffff6a479b4,%rbx            # mov the handler address into %rbx
callq   *%rbx                           # call handler via %rbx
leaveq                                  # mov rbp, rsp; pop rbp
pop     %rbx                            # restore rbx
pop     %rdi                            # restore rdi
pop     %rax                            # restore rax
jmpq    0x437a1f <gc_sweep+1096>        # continue execution
```

shortened assembly stub



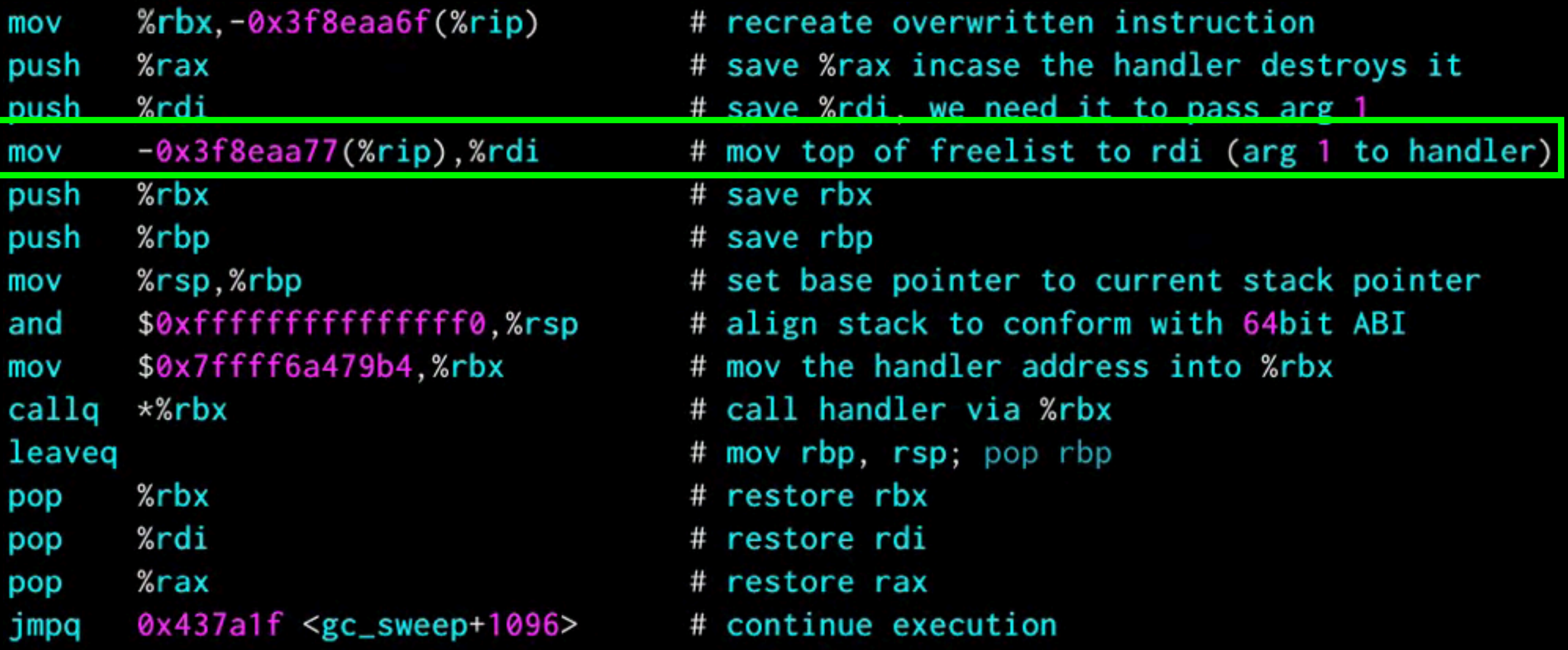
```
mov     %rbx,-0x3f8eaa6f(%rip)          # recreate overwritten instruction
push    %rax                            # save %rax incase the handler destroys it
push    %rdi                            # save %rdi, we need it to pass arg 1
mov     -0x3f8eaa77(%rip),%rdi          # mov top of freelist to rdi (arg 1 to handler)
push    %rbx                            # save rbx
push    %rbp                            # save rbp
mov     %rsp,%rbp                       # set base pointer to current stack pointer
and     $0xfffffffffffffff0,%rsp        # align stack to conform with 64bit ABI
mov     $0x7ffff6a479b4,%rbx            # mov the handler address into %rbx
callq   *%rbx                           # call handler via %rbx
leaveq                                  # mov rbp, rsp; pop rbp
pop     %rbx                            # restore rbx
pop     %rdi                            # restore rdi
pop     %rax                            # restore rax
jmpq    0x437a1f <gc_sweep+1096>        # continue execution
```

shortened assembly stub



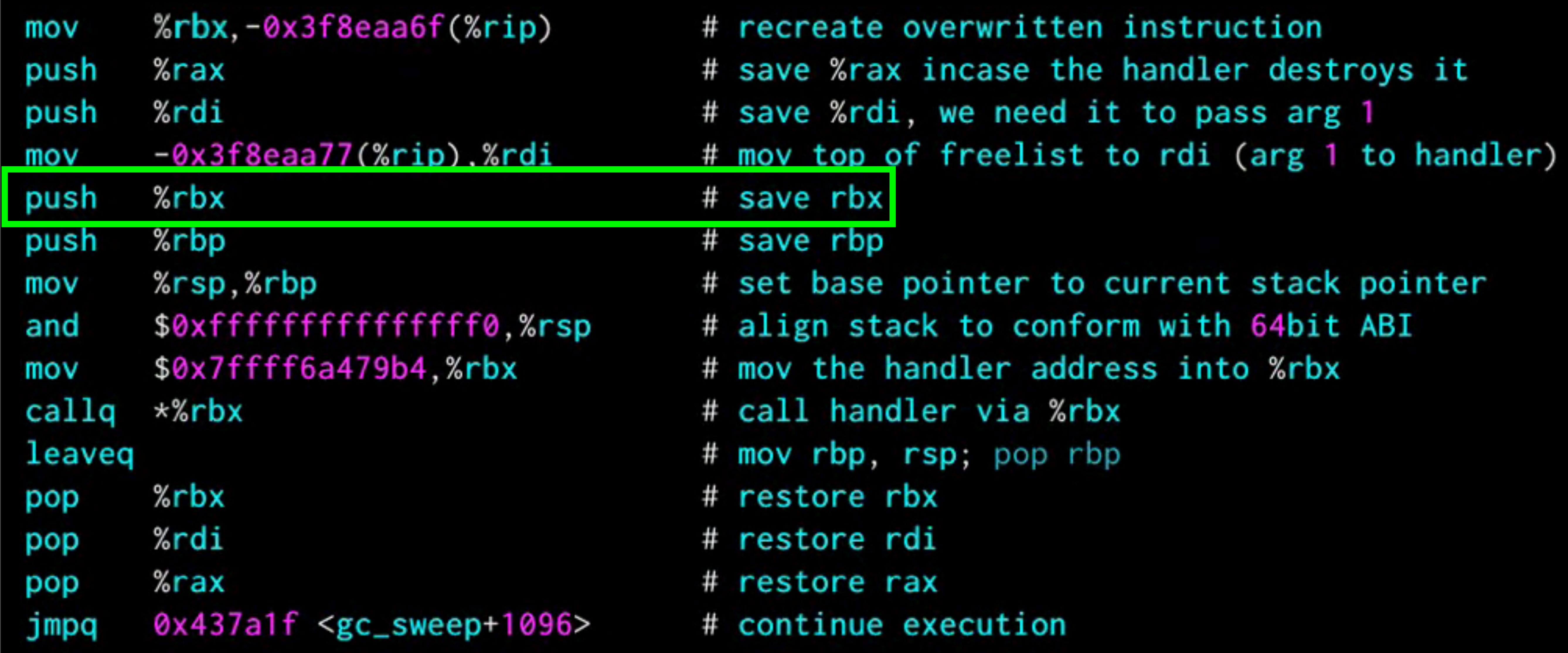
```
mov     %rbx,-0x3f8eaa6f(%rip)          # recreate overwritten instruction
push    %rax                            # save %rax incase the handler destroys it
push    %rdi                            # save %rdi, we need it to pass arg 1
mov     -0x3f8eaa77(%rip),%rdi          # mov top of freelist to rdi (arg 1 to handler)
push    %rbx                            # save rbx
push    %rbp                            # save rbp
mov     %rsp,%rbp                       # set base pointer to current stack pointer
and     $0xfffffffffffffff0,%rsp        # align stack to conform with 64bit ABI
mov     $0x7ffff6a479b4,%rbx            # mov the handler address into %rbx
callq   *%rbx                           # call handler via %rbx
leaveq                                  # mov rbp, rsp; pop rbp
pop     %rbx                            # restore rbx
pop     %rdi                            # restore rdi
pop     %rax                            # restore rax
jmpq    0x437a1f <gc_sweep+1096>        # continue execution
```

shortened assembly stub



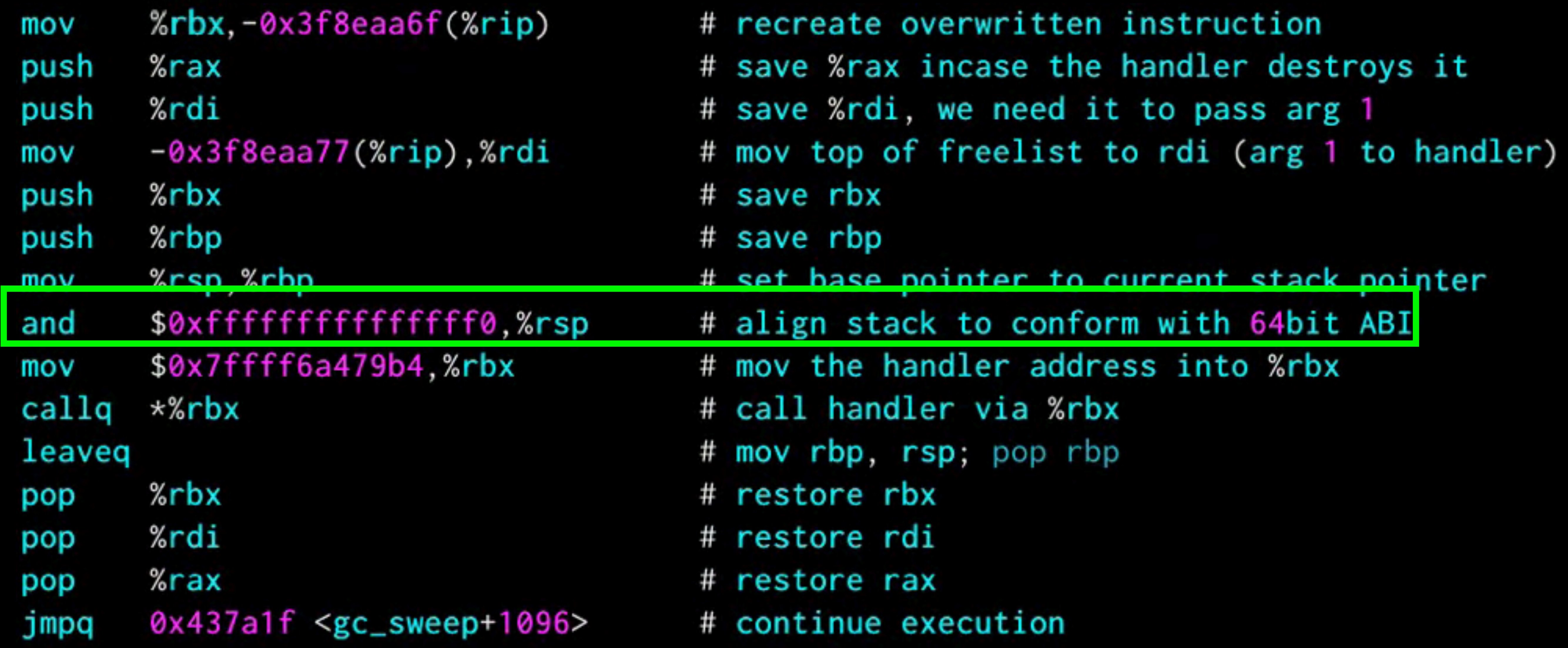
```
mov     %rbx,-0x3f8eaa6f(%rip)          # recreate overwritten instruction
push    %rax                            # save %rax incase the handler destroys it
push    %rdi                            # save %rdi, we need it to pass arg 1
mov     -0x3f8eaa77(%rip),%rdi          # mov top of freelist to rdi (arg 1 to handler)
push    %rbx                            # save rbx
push    %rbp                            # save rbp
mov     %rsp,%rbp                       # set base pointer to current stack pointer
and     $0xfffffffffffffff0,%rsp        # align stack to conform with 64bit ABI
mov     $0x7ffff6a479b4,%rbx            # mov the handler address into %rbx
callq   *%rbx                           # call handler via %rbx
leaveq                                  # mov rbp, rsp; pop rbp
pop     %rbx                            # restore rbx
pop     %rdi                            # restore rdi
pop     %rax                            # restore rax
jmpq    0x437a1f <gc_sweep+1096>        # continue execution
```

shortened assembly stub



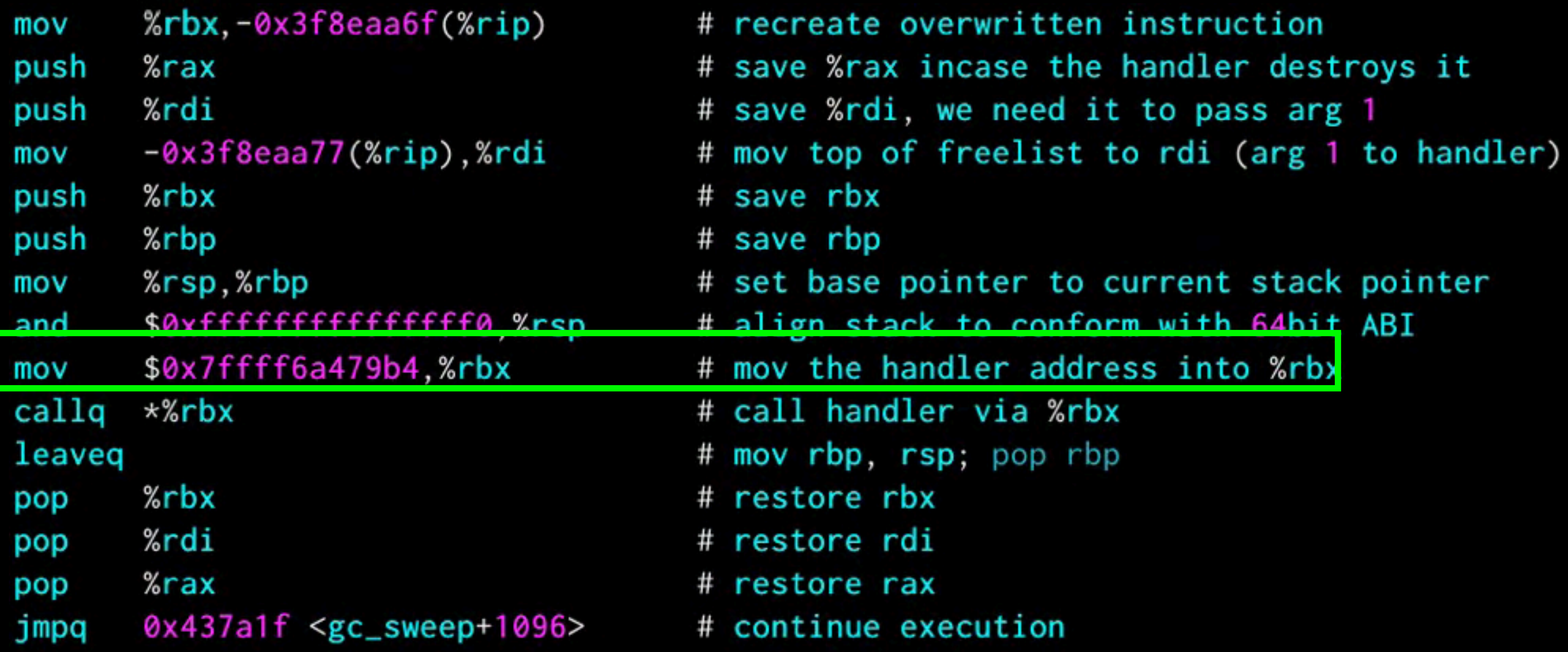
```
mov    %rbx,-0x3f8eaa6f(%rip)          # recreate overwritten instruction
push   %rax                            # save %rax incase the handler destroys it
push   %rdi                            # save %rdi, we need it to pass arg 1
mov    -0x3f8eaa77(%rip),%rdi          # mov top of freelist to rdi (arg 1 to handler)
push   %rbx                            # save rbx
push   %rbp                            # save rbp
mov    %rsp,%rbp                       # set base pointer to current stack pointer
and    $0xfffffffffffffff0,%rsp        # align stack to conform with 64bit ABI
mov    $0x7ffff6a479b4,%rbx            # mov the handler address into %rbx
callq  *%rbx                           # call handler via %rbx
leaveq                                 # mov rbp, rsp; pop rbp
pop    %rbx                            # restore rbx
pop    %rdi                            # restore rdi
pop    %rax                            # restore rax
jmpq   0x437a1f <gc_sweep+1096>        # continue execution
```

shortened assembly stub



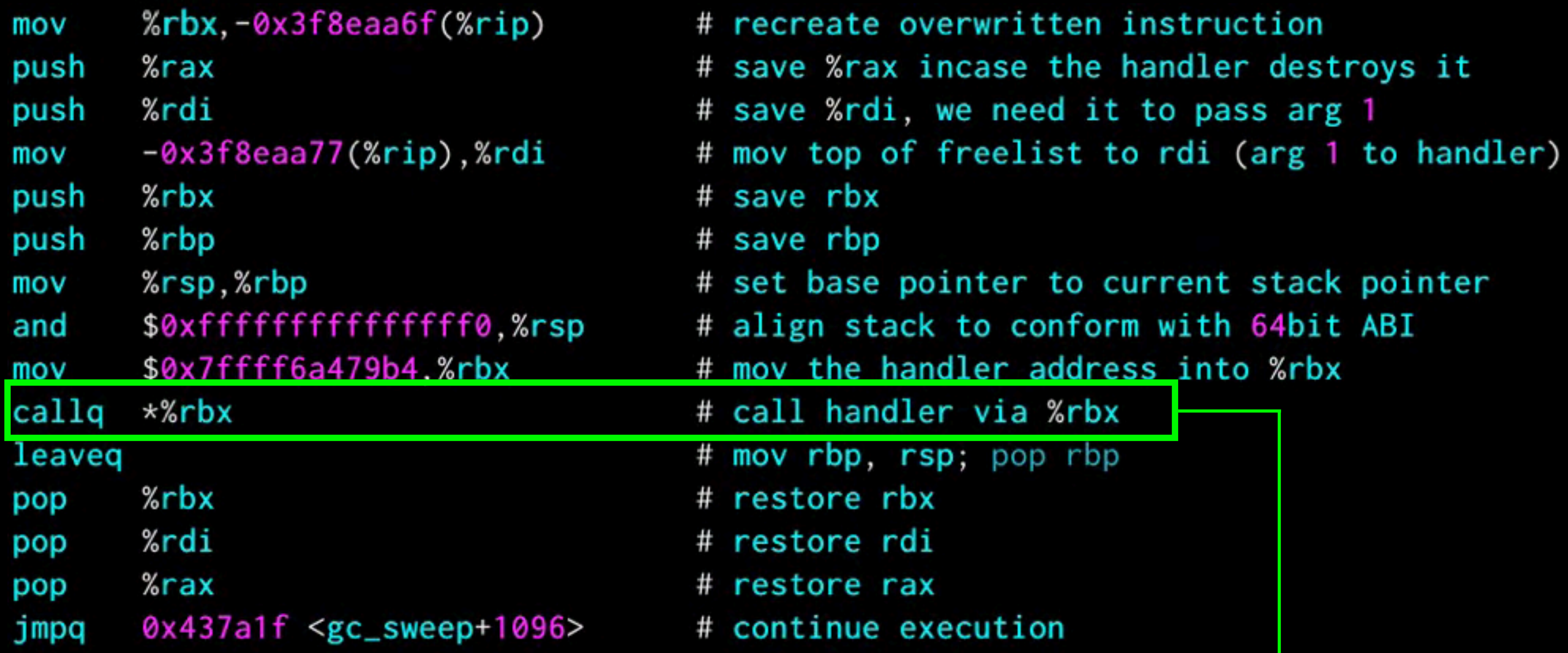
```
mov    %rbx,-0x3f8eaa6f(%rip)         # recreate overwritten instruction
push   %rax                           # save %rax incase the handler destroys it
push   %rdi                           # save %rdi, we need it to pass arg 1
mov    -0x3f8eaa77(%rip),%rdi         # mov top of freelist to rdi (arg 1 to handler)
push   %rbx                           # save rbx
push   %rbp                           # save rbp
mov    %rsp,%rbp                      # set base pointer to current stack pointer
and    $0xfffffffffffffff0,%rsp       # align stack to conform with 64bit ABI
mov    $0x7ffff6a479b4,%rbx           # mov the handler address into %rbx
callq  *%rbx                          # call handler via %rbx
leaveq                                # mov rbp, rsp; pop rbp
pop    %rbx                           # restore rbx
pop    %rdi                           # restore rdi
pop    %rax                           # restore rax
jmpq   0x437a1f <gc_sweep+1096>       # continue execution
```

shortened assembly stub



```
mov     %rbx,-0x3f8eaa6f(%rip)          # recreate overwritten instruction
push    %rax                            # save %rax incase the handler destroys it
push    %rdi                            # save %rdi, we need it to pass arg 1
mov     -0x3f8eaa77(%rip),%rdi          # mov top of freelist to rdi (arg 1 to handler)
push    %rbx                            # save rbx
push    %rbp                            # save rbp
mov     %rsp,%rbp                       # set base pointer to current stack pointer
and     $0xfffffffffffffff0,%rsp        # align stack to conform with 64bit ABI
mov     $0x7ffff6a479b4,%rbx            # mov the handler address into %rbx
callq   *%rbx                           # call handler via %rbx
leaveq                                  # mov rbp, rsp; pop rbp
pop     %rbx                            # restore rbx
pop     %rdi                            # restore rdi
pop     %rax                            # restore rax
jmpq    0x437a1f <gc_sweep+1096>        # continue execution
```

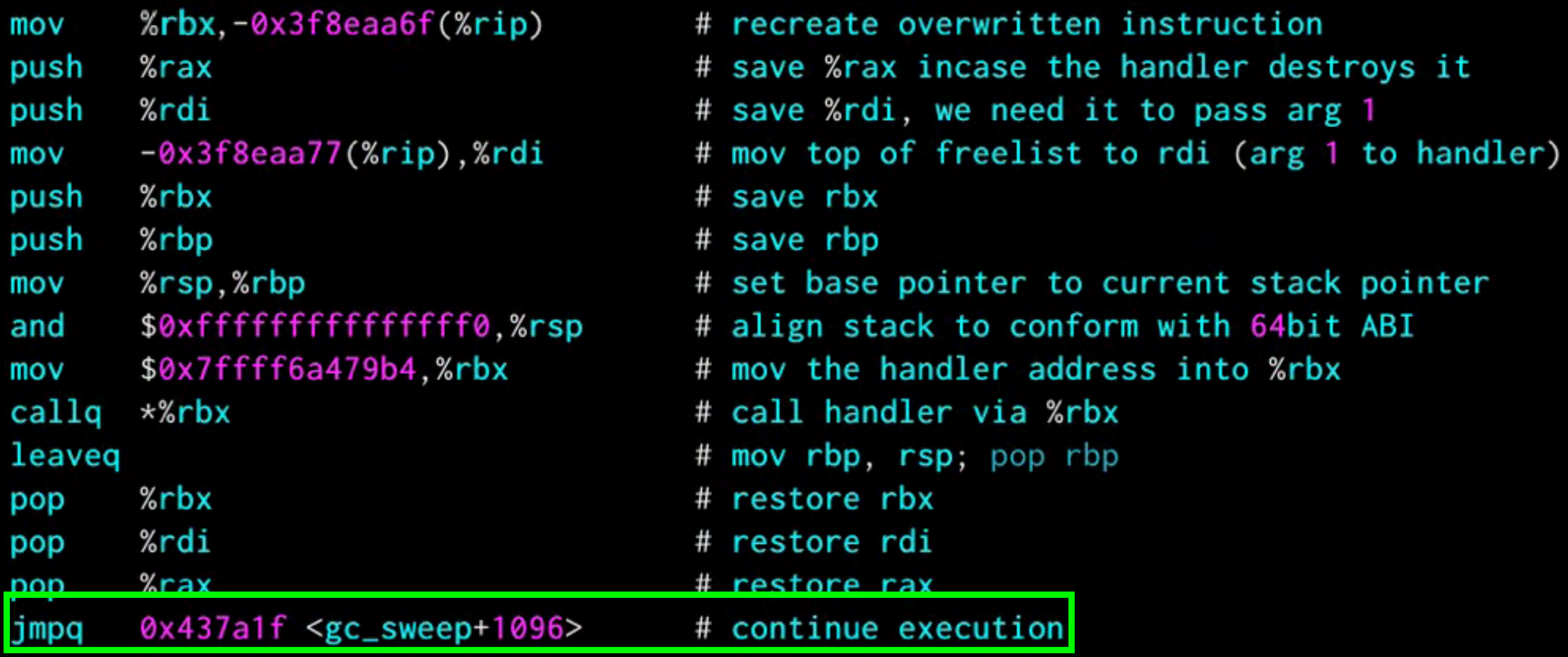
shortened assembly stub



```
mov     %rbx,-0x3f8eaa6f(%rip)        # recreate overwritten instruction
push    %rax                          # save %rax incase the handler destroys it
push    %rdi                          # save %rdi, we need it to pass arg 1
mov     -0x3f8eaa77(%rip),%rdi        # mov top of freelist to rdi (arg 1 to handler)
push    %rbx                          # save rbx
push    %rbp                          # save rbp
mov     %rsp,%rbp                     # set base pointer to current stack pointer
and     $0xfffffffffffffff0,%rsp      # align stack to conform with 64bit ABI
mov     $0x7ffff6a479b4,%rbx          # mov the handler address into %rbx
callq   *%rbx                         # call handler via %rbx
leaveq                                # mov rbp, rsp; pop rbp
pop     %rbx                          # restore rbx
pop     %rdi                          # restore rdi
pop     %rax                          # restore rax
jmpq    0x437a1f <gc_sweep+1096>      # continue execution
```

```
void handler(VALUE freed_object)
{
        mark_object_freed(freed_object);
        return;
}
```

shortened assembly stub



```
mov     %rbx,-0x3f8eaa6f(%rip)      # recreate overwritten instruction
push    %rax                        # save %rax incase the handler destroys it
push    %rdi                        # save %rdi, we need it to pass arg 1
mov     -0x3f8eaa77(%rip),%rdi      # mov top of freelist to rdi (arg 1 to handler)
push    %rbx                        # save rbx
push    %rbp                        # save rbp
mov     %rsp,%rbp                   # set base pointer to current stack pointer
and     $0xfffffffffffffff0,%rsp    # align stack to conform with 64bit ABI
mov     $0x7ffff6a479b4,%rbx        # mov the handler address into %rbx
callq   *%rbx                       # call handler via %rbx
leaveq                              # mov rbp, rsp; pop rbp
pop     %rbx                        # restore rbx
pop     %rdi                        # restore rdi
pop     %rax                        # restore rax
jmpq    0x437a1f <gc_sweep+1096>    # continue execution
```

and it actually works.

```
gem install memprof
```

http://github.com/ice799/memprof

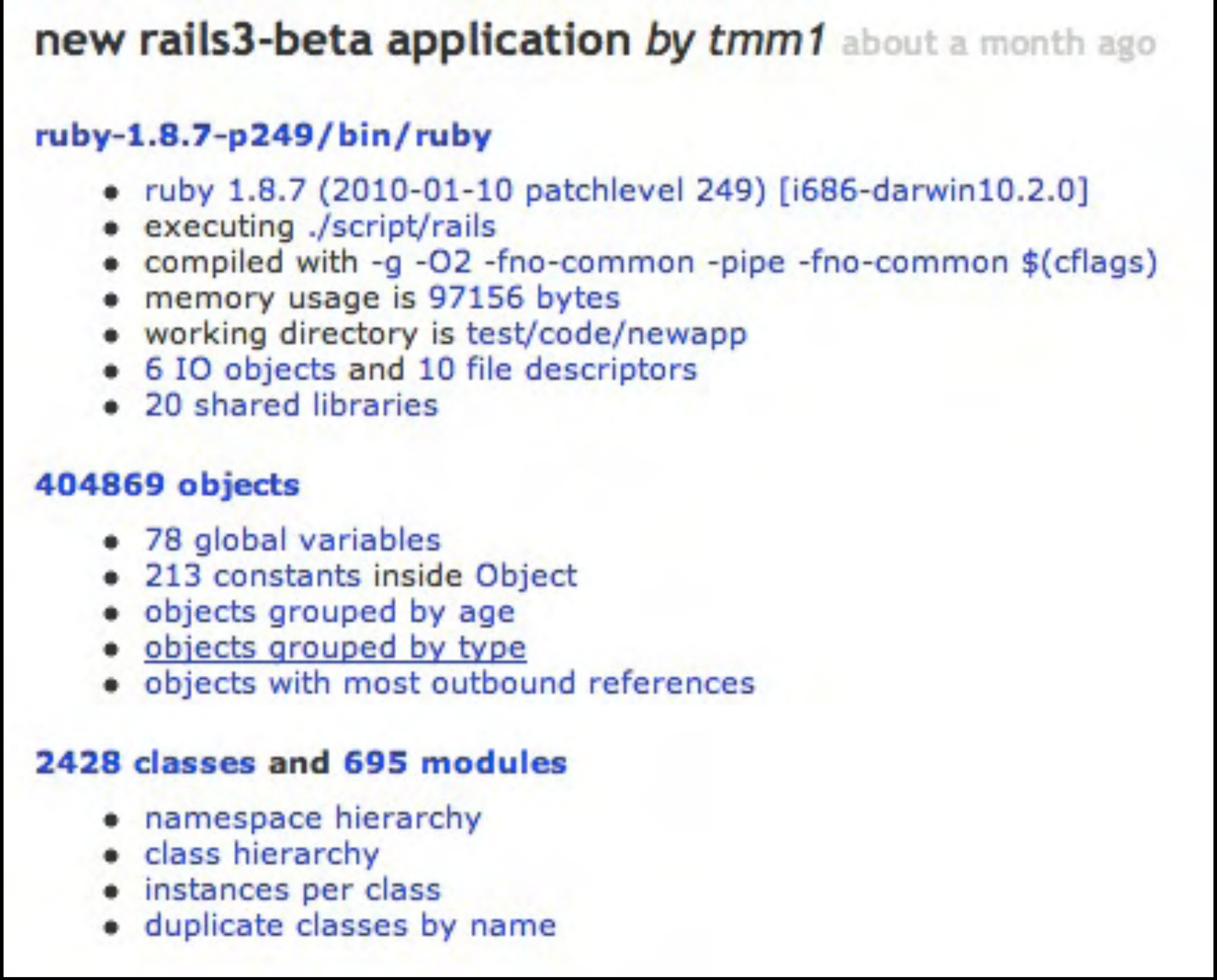
Sample Output

```
require 'memprof'
Memprof.start
require "stringio"
StringIO new
Memprof.stats
```

```
108 /custom/ree/lib/ruby/1.8/x86_64-linux/stringio.so:0:__node__
 14 test2.rb:3:String
  2 /custom/ree/lib/ruby/1.8/x86_64-linux/stringio.so:0:Class
  1 test2.rb:4:StringIO
  1 test2.rb:4:String
  1 test2.rb:3:Array
  1 /custom/ree/lib/ruby/1.8/x86_64-linux/stringio.so:0:Enumerable
```

memprof.com

a web-based heap visualizer and leak analyzer



new rails3-beta application *by tmm1* about a month ago

ruby-1.8.7-p249/bin/ruby

- ruby 1.8.7 (2010-01-10 patchlevel 249) [i686-darwin10.2.0]
- executing ./script/rails
- compiled with -g -O2 -fno-common -pipe -fno-common $(cflags)
- memory usage is 97156 bytes
- working directory is test/code/newapp
- 6 IO objects and 10 file descriptors
- 20 shared libraries

404869 objects

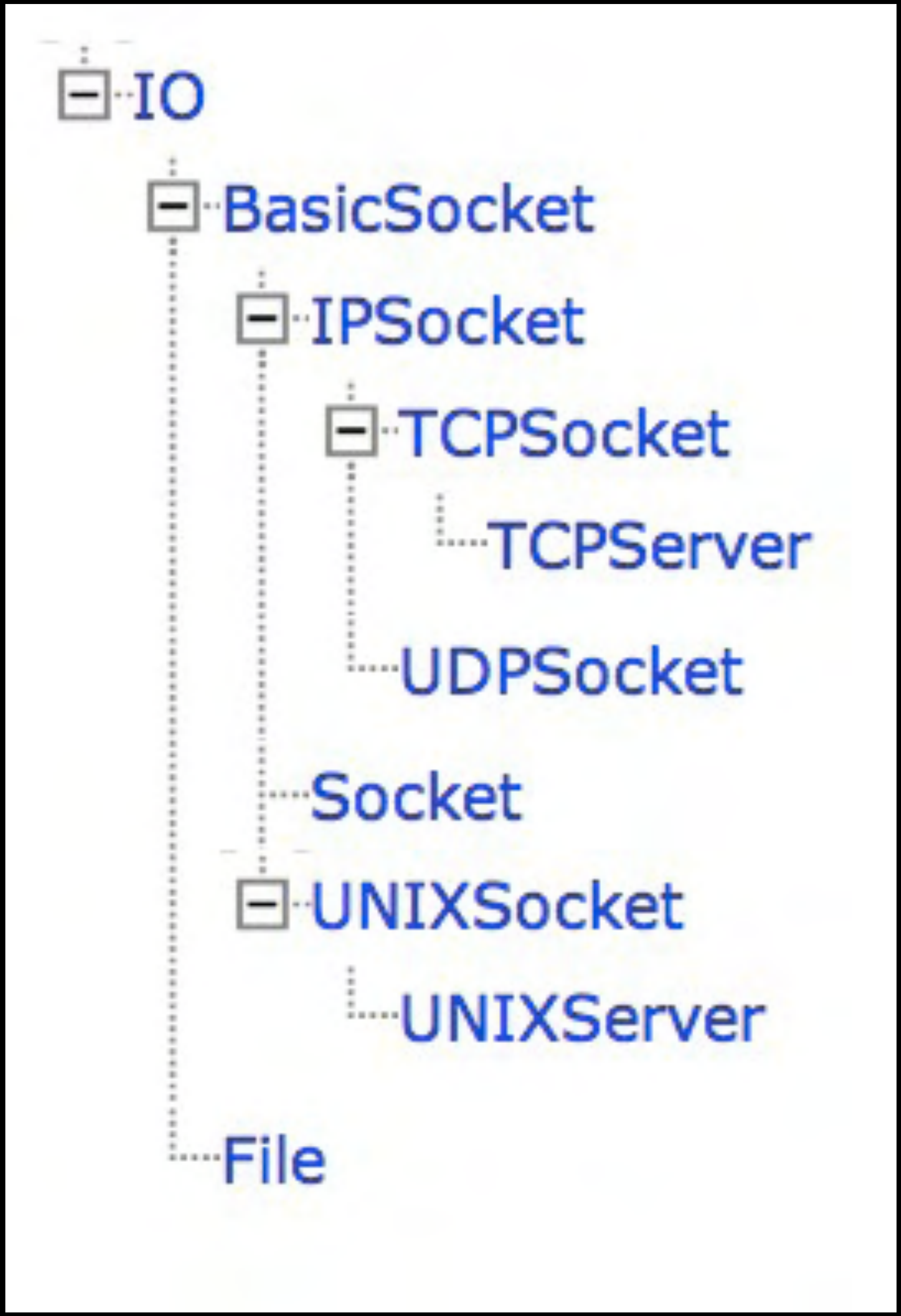
- 78 global variables
- 213 constants inside Object
- objects grouped by age
- objects grouped by type
- objects with most outbound references

2428 classes and 695 modules

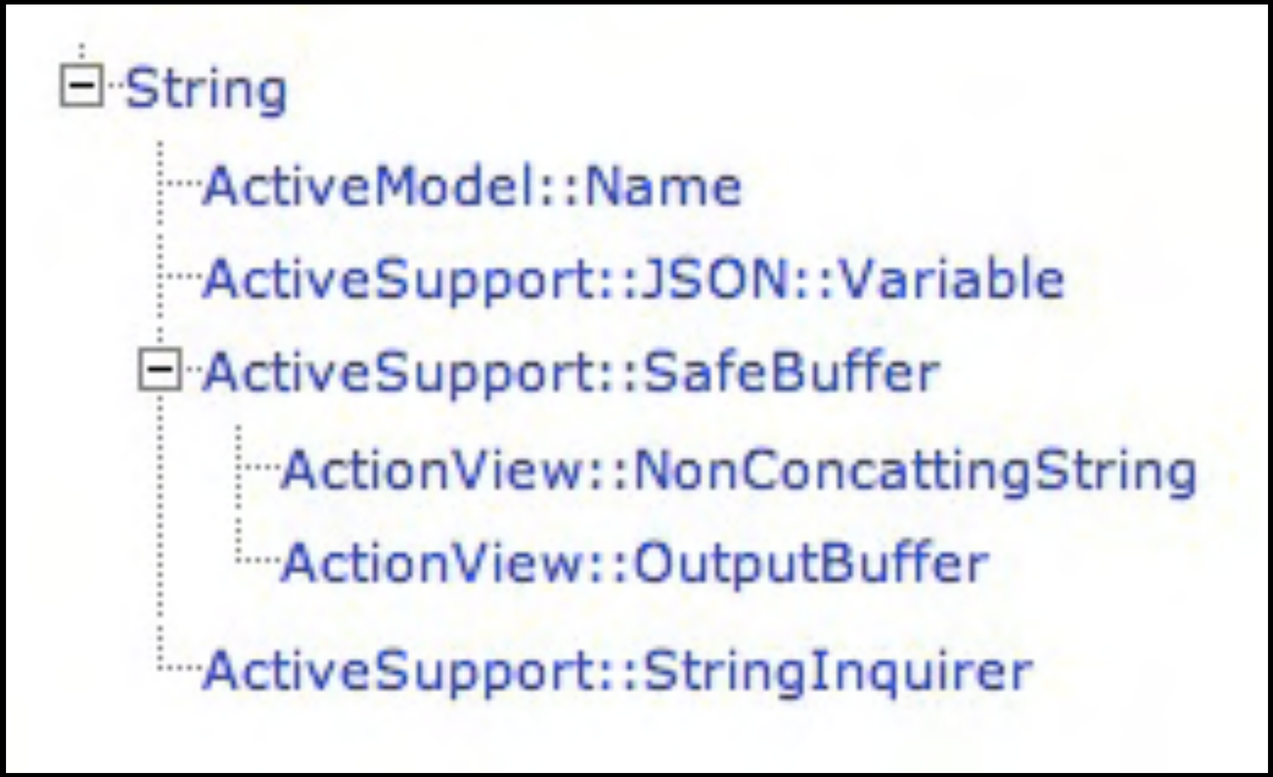
- namespace hierarchy
- class hierarchy
- instances per class
- duplicate classes by name

memprof.com

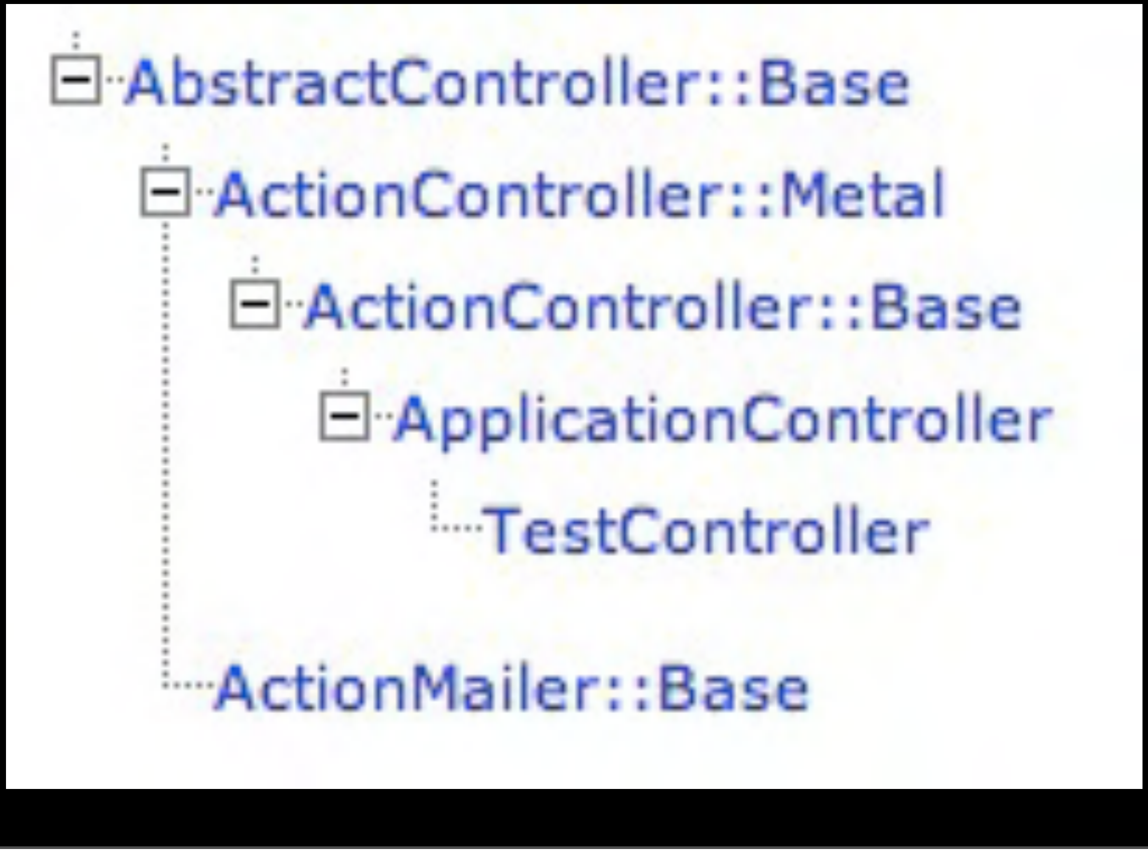
a web-based heap visualizer and leak analyzer



- IO
  - BasicSocket
    - IPSocket
      - TCPSocket
        - TCPServer
      - UDPSocket
    - Socket
    - UNIXSocket
      - UNIXServer
  - File



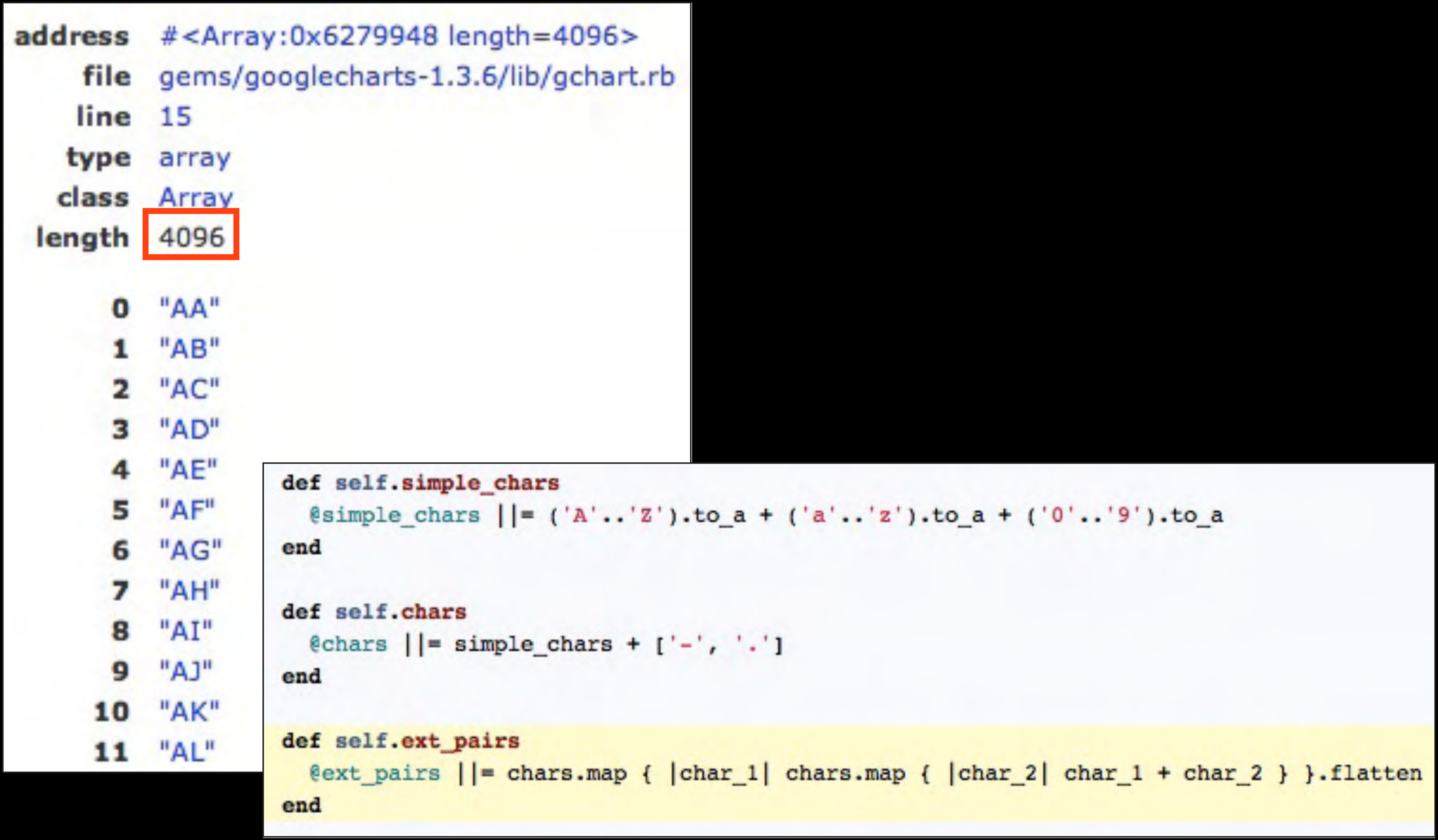
- String
  - ActiveModel::Name
  - ActiveSupport::JSON::Variable
  - ActiveSupport::SafeBuffer
    - ActionView::NonConcattingString
    - ActionView::OutputBuffer
  - ActiveSupport::StringInquirer



- AbstractController::Base
  - ActionController::Metal
    - ActionController::Base
      - ApplicationController
        - TestController
  - ActionMailer::Base

memprof.com

a web-based heap visualizer and leak analyzer

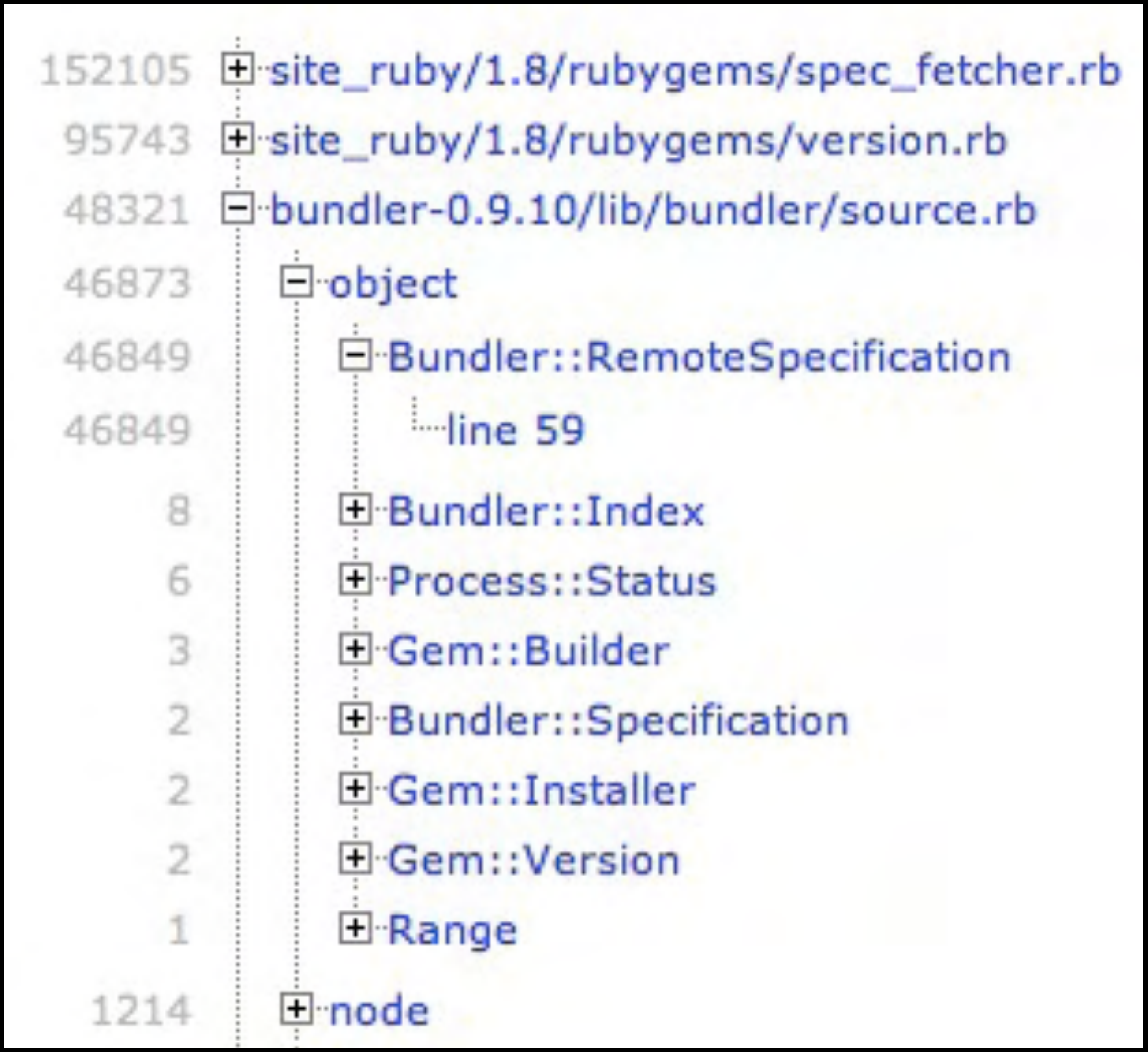


address	#<Array:0x6279948 length=4096>
file	gems/googlecharts-1.3.6/lib/gchart.rb
line	15
type	array
class	Array
length	4096
0	"AA"
1	"AB"
2	"AC"
3	"AD"
4	"AE"
5	"AF"
6	"AG"
7	"AH"
8	"AI"
9	"AJ"
10	"AK"
11	"AL"

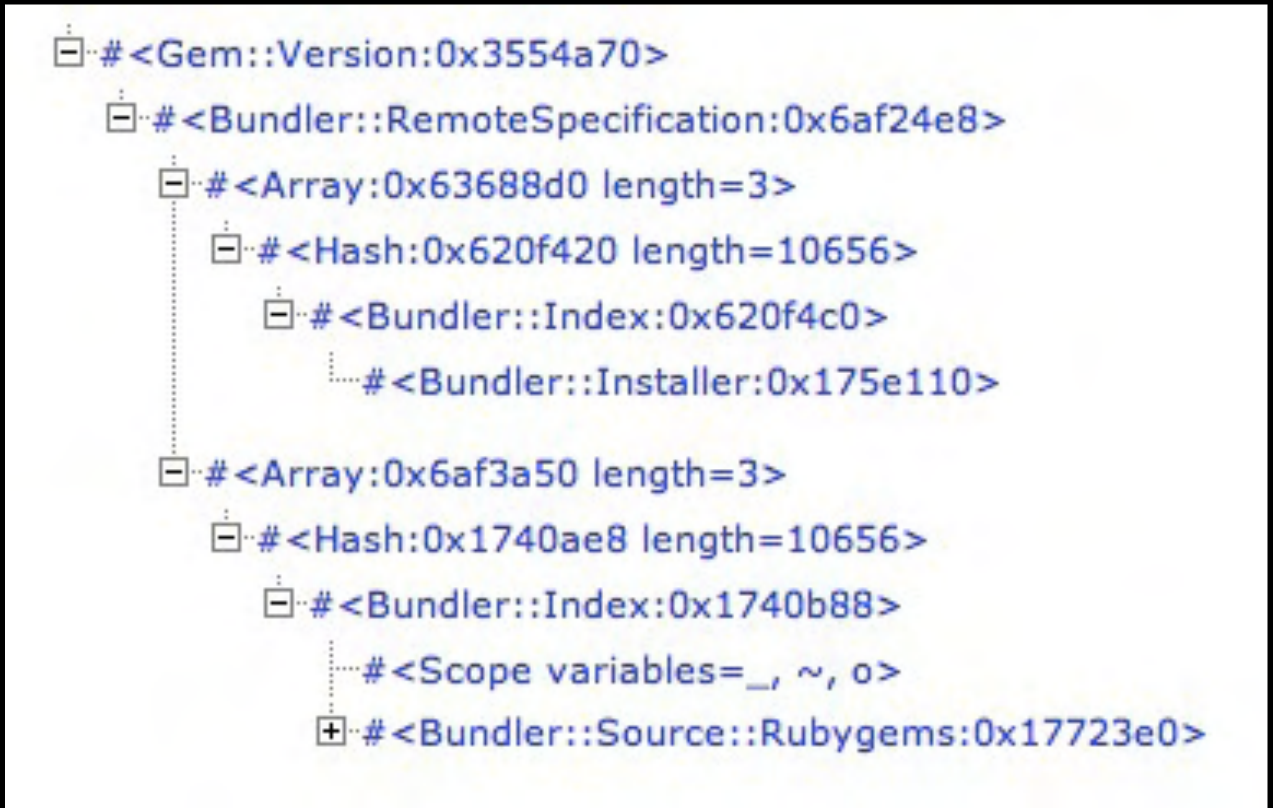
```
def self.simple_chars
  @simple_chars ||= ('A'..'Z').to_a + ('a'..'z').to_a + ('0'..'9').to_a
end

def self.chars
  @chars ||= simple_chars + ['-', '.']
end

def self.ext_pairs
  @ext_pairs ||= chars.map { |char_1| chars.map { |char_2| char_1 + char_2 } }.flatten
end
```

memprof.com

a web-based heap visualizer and leak analyzer



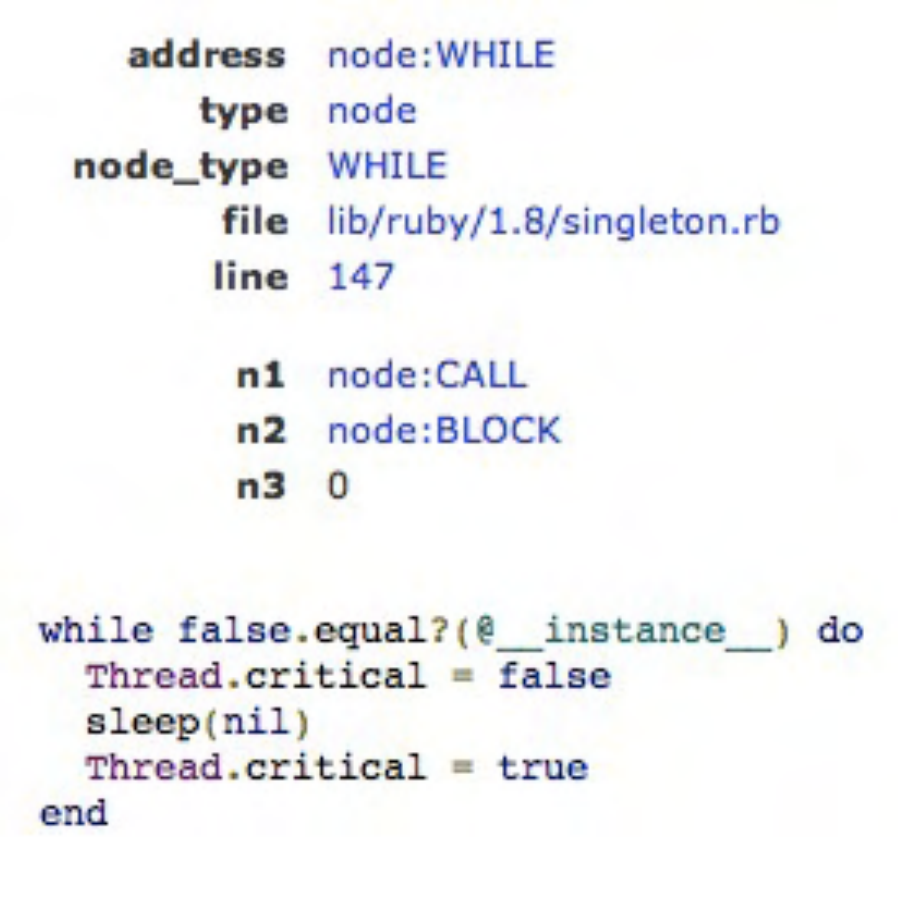
```
152105 ⊞ site_ruby/1.8/rubygems/spec_fetcher.rb
 95743 ⊞ site_ruby/1.8/rubygems/version.rb
 48321 ⊟ bundler-0.9.10/lib/bundler/source.rb
 46873     ⊟ object
 46849         ⊟ Bundler::RemoteSpecification
 46849             line 59
     8         ⊞ Bundler::Index
     6         ⊞ Process::Status
     3         ⊞ Gem::Builder
     2         ⊞ Bundler::Specification
     2         ⊞ Gem::Installer
     2         ⊞ Gem::Version
     1         ⊞ Range
  1214     ⊞ node
```



```
⊟ #<Gem::Version:0x3554a70>
  ⊟ #<Bundler::RemoteSpecification:0x6af24e8>
    ⊟ #<Array:0x63688d0 length=3>
      ⊟ #<Hash:0x620f420 length=10656>
        ⊟ #<Bundler::Index:0x620f4c0>
            #<Bundler::Installer:0x175e110>
    ⊟ #<Array:0x6af3a50 length=3>
      ⊟ #<Hash:0x1740ae8 length=10656>
        ⊟ #<Bundler::Index:0x1740b88>
            #<Scope variables=_, ~, o>
          ⊞ #<Bundler::Source::Rubygems:0x17723e0>
```

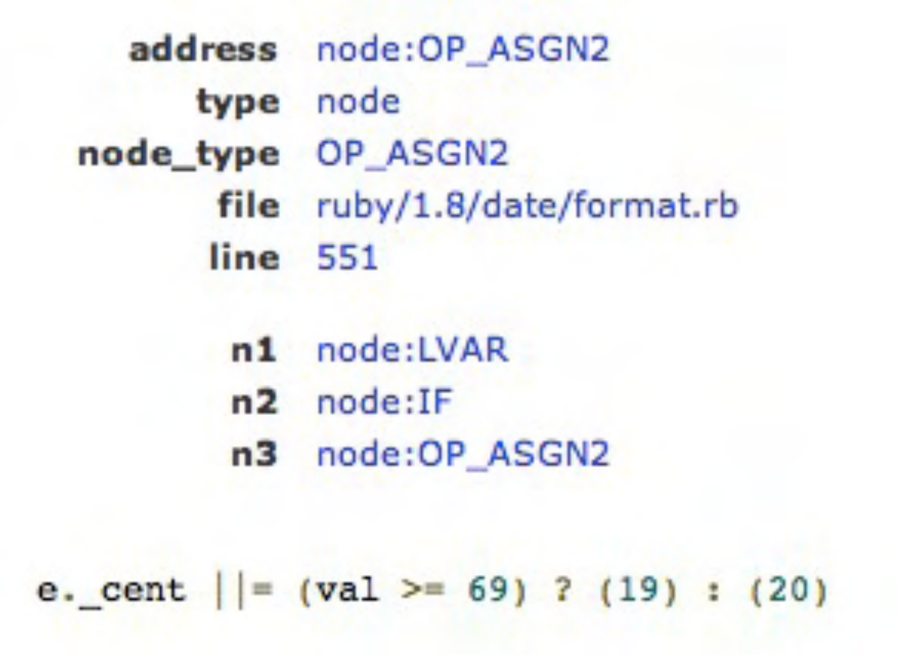
memprof.com

a web-based heap visualizer and leak analyzer



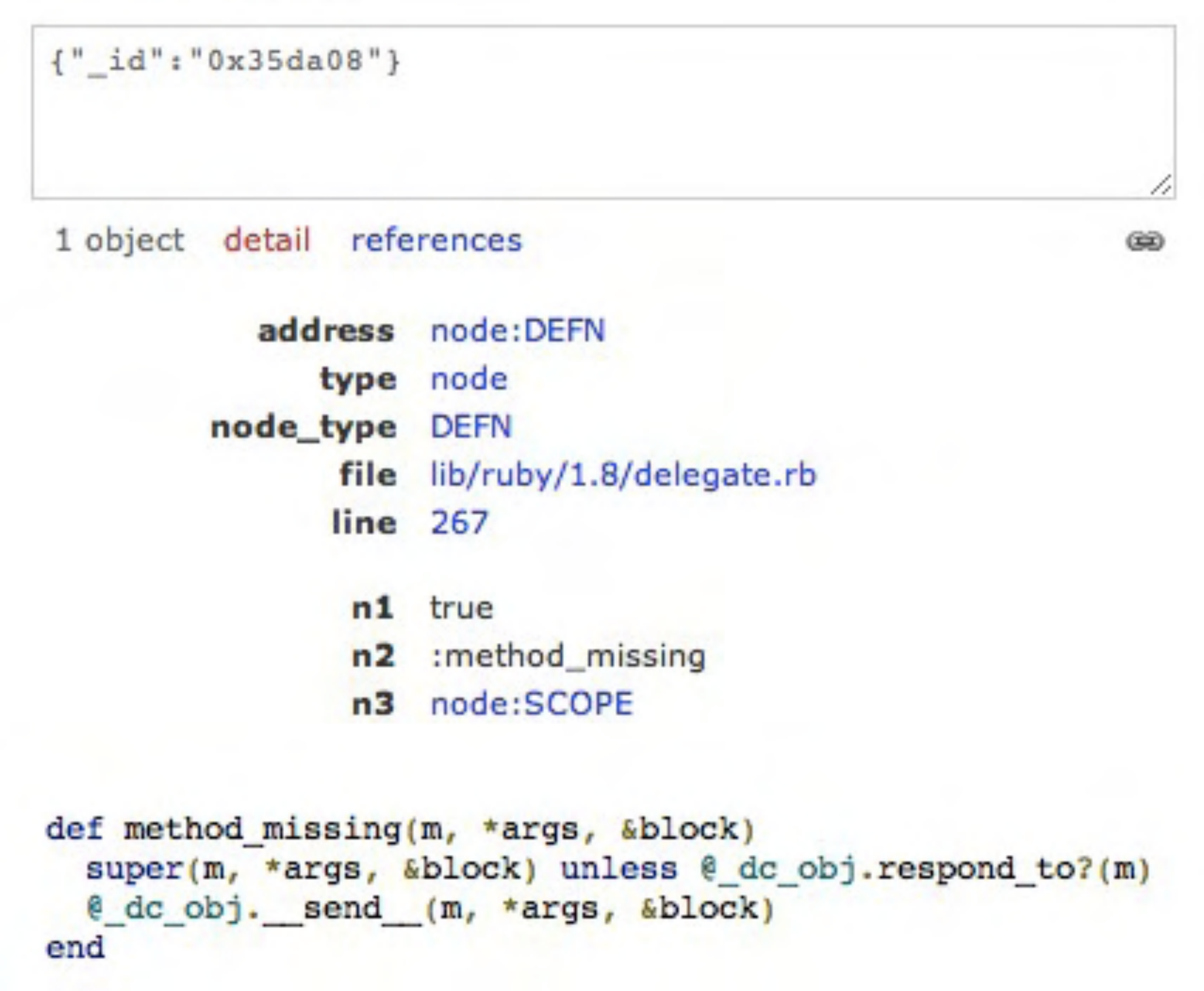
address	node:WHILE
type	node
node_type	WHILE
file	lib/ruby/1.8/singleton.rb
line	147
n1	node:CALL
n2	node:BLOCK
n3	0

```
while false.equal?(@__instance__) do
  Thread.critical = false
  sleep(nil)
  Thread.critical = true
end
```



address	node:OP_ASGN2
type	node
node_type	OP_ASGN2
file	ruby/1.8/date/format.rb
line	551
n1	node:LVAR
n2	node:IF
n3	node:OP_ASGN2

```
e._cent ||= (val >= 69) ? (19) : (20)
```



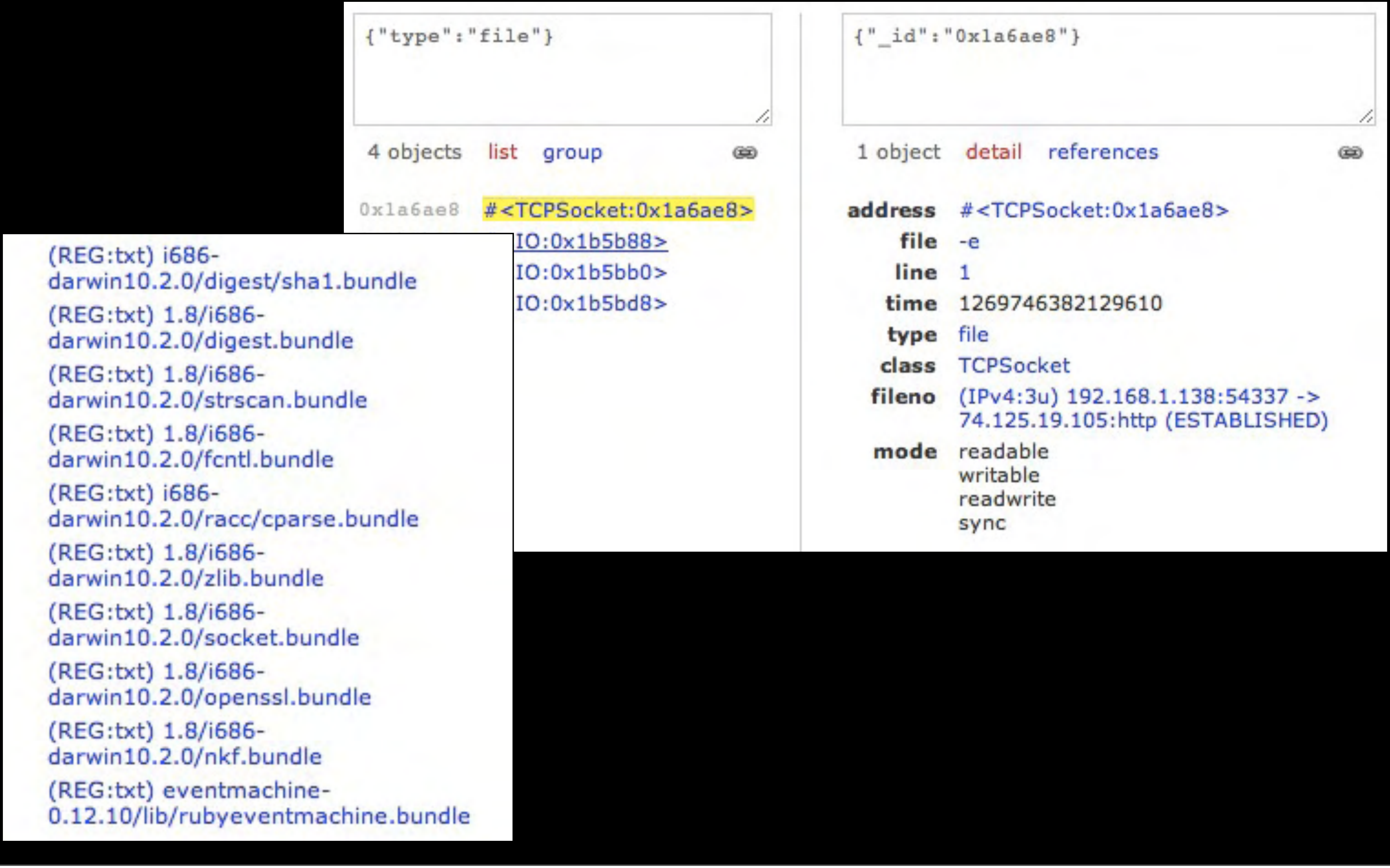
```
{"_id":"0x35da08"}
```

address	node:DEFN
type	node
node_type	DEFN
file	lib/ruby/1.8/delegate.rb
line	267
n1	true
n2	:method_missing
n3	node:SCOPE

```
def method_missing(m, *args, &block)
  super(m, *args, &block) unless @_dc_obj.respond_to?(m)
  @_dc_obj.__send__(m, *args, &block)
end
```

memprof.com

a web-based heap visualizer and leak analyzer



```
{"type":"file"}
```

4 objects list group

0x1a6ae8 #<TCPSocket:0x1a6ae8>

IO:0x1b5b88>

IO:0x1b5bb0>

IO:0x1b5bd8>

```
{"_id":"0x1a6ae8"}
```

1 object detail references

address	#<TCPSocket:0x1a6ae8>
file	-e
line	1
time	1269746382129610
type	file
class	TCPSocket
fileno	(IPv4:3u) 192.168.1.138:54337 -> 74.125.19.105:http (ESTABLISHED)
mode	readable writable readwrite sync

(REG:txt) i686-darwin10.2.0/digest/sha1.bundle

(REG:txt) 1.8/i686-darwin10.2.0/digest.bundle

(REG:txt) 1.8/i686-darwin10.2.0/strscan.bundle

(REG:txt) 1.8/i686-darwin10.2.0/fcntl.bundle

(REG:txt) i686-darwin10.2.0/racc/cparse.bundle

(REG:txt) 1.8/i686-darwin10.2.0/zlib.bundle

(REG:txt) 1.8/i686-darwin10.2.0/socket.bundle

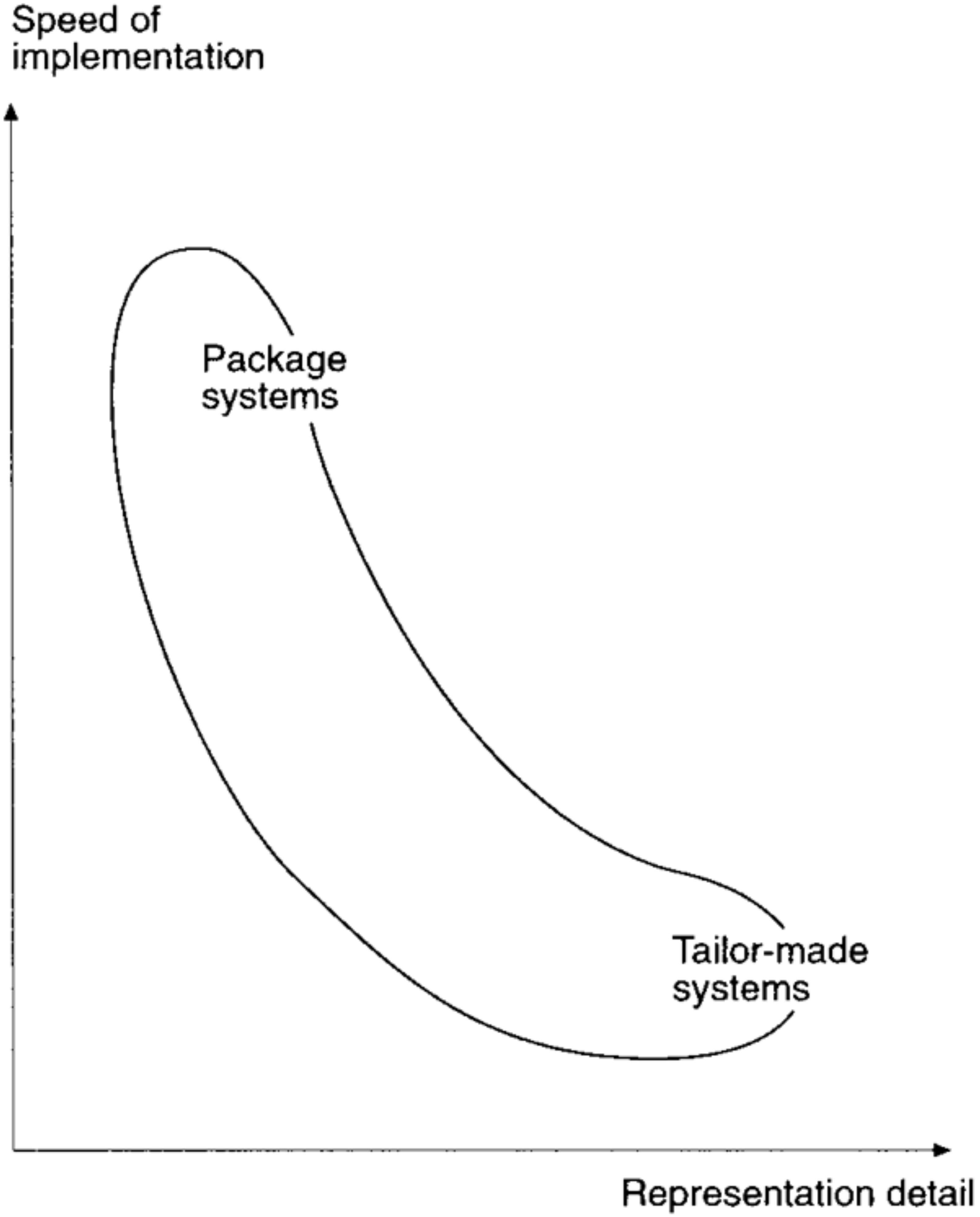
(REG:txt) 1.8/i686-darwin10.2.0/openssl.bundle

(REG:txt) 1.8/i686-darwin10.2.0/nkf.bundle

(REG:txt) eventmachine-0.12.10/lib/rubyeventmachine.bundle

evilgem demo/example?

buycostumes.com



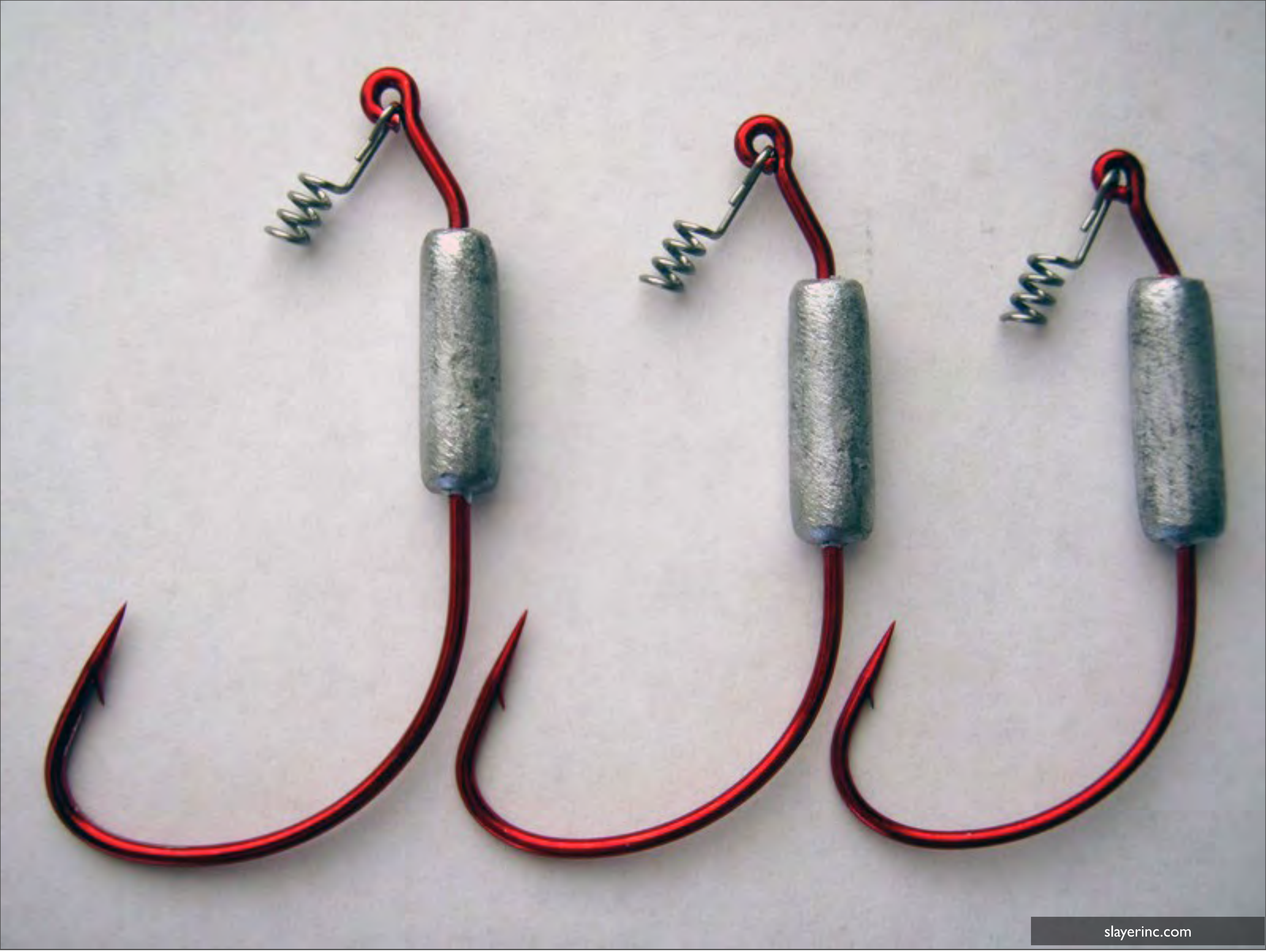
emeraldinsight.com

how to defend against it

- NX bit - call mprotect
- strip debug information - mostly prebuilt binaries
- statically link everything - extremely large binaries
- put all .text code in ROM - maybe?
- don’t load DSOs at runtime. - no plugins, though

EXPERIMENTAL
SEGWAY
GM
EXPERIMENTAL
slashgear.com

my future research: exploring alternative binary formats.



slayerinc.com

globalhealthandfitness.com

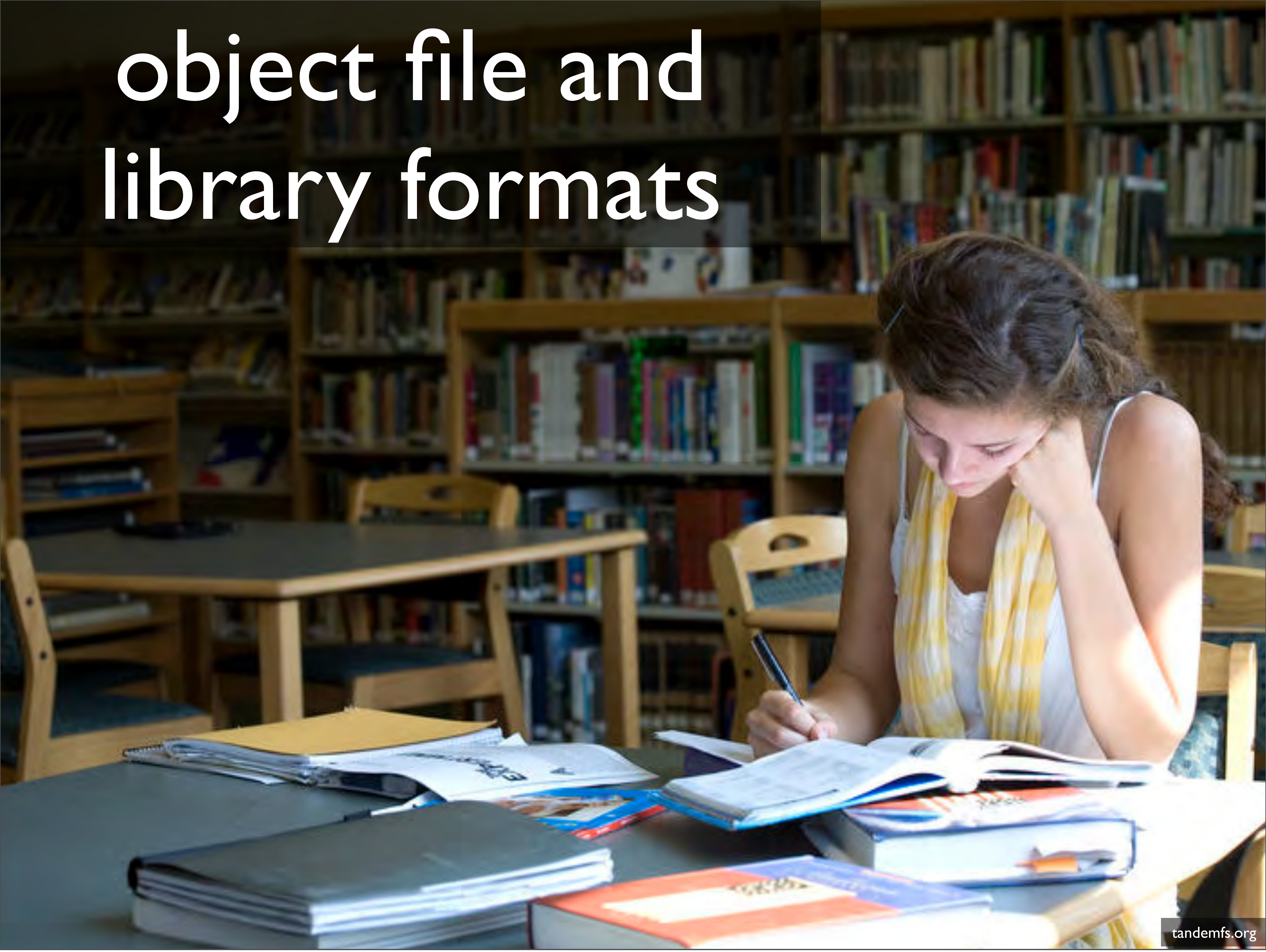
alignment

thomasgroup.com

calling convention

arianlim.wordpress.com

object file and library formats



tandemfs.org

questions?

joe damato
@joedamato
timetobleed.com

read more:

http://timetobleed.com/string-together-global-offset-tables-to-build-a-ruby-memory-profiler/
http://timetobleed.com/hot-patching-inlined-functions-with-x86_64-asm-metaprogramming/
http://timetobleed.com/rewrite-your-ruby-vm-at-runtime-to-hot-patch-useful-features/